December 1987

UILU-ENG-87-2275
CSG-79

COORDINATED SCIENCE LABORATORY
College of Engineering

FILE USAGE ANALYSIS AND RESOURCE USAGE PREDICTION: A MEASUREMENT-BASED STUDY

Murthy V.–S. Devarakonda

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

FILE USAGE ANALYSIS AND RESOURCE USAGE PREDICTION: A MEASUREMENT-BASED STUDY

BY

MURTHY V.-S. DEVARAKONDA

B.E., Osmania University, 1978
M.Tech., Indian Institute of Technology, 1980
M.S., University of Wisconsin-Madison, 1983

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1988

Urbana, Illinois

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

October 15, 1987

WE HEREBY RECOMMEND THAT THE THESIS BY

MURTHY V.-S. DEVARAKONDA

ENTITLED File Usage Analysis and Resource Usage Prediction:

A Measurement-Based Study

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF Doctor of Philosophy

R. K. Iyer
Director of Thesis Research

M. Faiman
Head of Department

Committee on Final Examination†

R. K. Iyer
Chairperson

Janak H. Patel

Duncan

W. K. Fuchs

Ravi Campbell

† Required for doctor's degree but not for master's.

O-517

ABSTRACT

In this paper, a probabilistic scheme was developed to predict process resource usage in UNIX. Given the identity of the program being run, the scheme predicts CPU time, file I/O, and memory requirements of a process at the beginning of its life. The scheme uses a state-transition model of the program's resource usage in its past executions for prediction. The states of the model are the resource regions obtained from an off-line cluster analysis of processes run on the system. The proposed method is shown to work on data collected from a VAX 11/780 running 4.3 BSD UNIX. The results show that the predicted values correlate well with the actual. The coefficient of correlation between the predicted and actual values of CPU time is 0.84. Errors in prediction are mostly small. About 82% of errors in CPU time prediction are less than 0.5 standard deviations of process CPU time.

FILE USAGE ANALYSIS AND RESOURCE USAGE PREDICTION: A MEASUREMENT-BASED STUDY

Murthy Devarakonda, Ph. D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1988
R. K. Iyer, Advisor

This thesis demonstrates a practical methodology for file usage analysis and resource usage prediction using trace-data from a production system. A VAX 11/780 system running Berkeley UNIX (Version 4.2 first, and 4.3 later), was instrumented to gather file usage data, in the form of file-related system calls, and resource usage data, such as CPU time and memory usage, for each process. The data was collected on a continuous trace basis in two sets of measurements.

First, a user-oriented analysis was done using the file usage data collected from the first measurement. The key aspect of this analysis is a characterization of users and files. Two characterization measures are employed: accesses-per-byte (that combines fraction of a file referenced and number of references) and file size. This new approach is shown to distinguish differences in files as well as in users, which can be used in efficient file system design, and in creating realistic test workloads for simulations. A multi-stage gamma distribution is shown to closely model the file usage measures. Even though overall file sharing is small, some files belonging to a bulletin board system are accessed by many users, simultaneously and otherwise. About 50% of users referenced files owned by other users, and over 8% of all files were involved in such references. Based on the differences in files and users, suggestions to improve file system performance were also made.

Next, the file usage data from the second measurement is analyzed using a few simple measures based on the notion of a file reference. A file reference starts with an *open* or a *creat* call to a file, encompasses any subsequent *reads*, *writes*, or *lseeks*, and concludes with an explicit *close* system call or termination of the process that started the reference. The measures used are: fraction referenced, file size, reference-time, number of references, and inter-reference time. Neither the users nor the files were

characterized in this analysis. Results from this analysis are seen to complement those obtained from that of the user-oriented analysis. It was shown that in most references, files were accessed completely (if accessed at all), substantiating the argument for using access-per-byte measure in user-oriented analysis. It was also shown that most file references lasted for a short time (median: 0.08 seconds), and that inter-reference time was 2 to 3 orders of magnitude larger (median: 45 seconds) than reference time.

Finally, a probabilistic resource usage prediction scheme was developed, using the process resource usage data. Given the identity of the program being run, the scheme predicts CPU time, file I/O, and memory requirements of a process at the beginning of its life. The scheme uses a state-transition model of a program's resource usage in its past executions for prediction. The states of the model are the resource regions obtained from an off-line cluster analysis of processes run on the system. The proposed method is shown to work on data collected from a VAX 11/780 running 4.3 BSD UNIX. The results show that the predicted values correlate well with the actual; coefficient of correlation for CPU time is 0.84. Errors in prediction are mostly small; 82% of errors in CPU time prediction are less than 0.5 standard deviations of process CPU time.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my thesis advisor, Professor Ravi Iyer. He spent many long hours discussing the research and putting together this thesis. If I have learned a little about measurement-based research and how to interpret the results, it is a direct consequence of these extensive interactions with Professor Iyer. At a personal level, his guidance and friendship have been equally invaluable.

I am also indebted to Professor Roy Campbell, whose interest in this work has been unwavering. He has been a constant source of encouragement. As a member of Professor Campbell's research group, I have learned much about the UNIX operating system and computer networks. It is with pleasure that I acknowledge the opportunities he provided me.

I would like to thank the rest of my thesis committee, Professors Fuchs, Lawrie, Patel, and Reed, for their comments and suggestions.

I thank all the members, past and present, staff and students, of Professional Workstation Group and Computer Systems Group for a rewarding experience and an enjoyable atmosphere. In particular, I owe thanks to Luke Young, Rick Eichemeyer, and Sharon Peterson for making this thesis readable, and to Rene Llames, Mark Sloan, and Kumar Goswami for interesting discussions.

Thanks are to my friends here at Champaign-Urbana, and in Madison. Their friendships have been very important to me. In particular, I thank Prasanna for helping me keep my sanity.

Finally, my parents, brothers, and my grand mother endured endless anxiety and long separation while I pursued a higher education abroad. For all their love and support, I am grateful.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

File access performance and process scheduling are the two key aspects of computer system design that influence the performance of a computer system. File access time, i.e., the time taken by an executing program to read from or write to a file, is considerably larger when the file is on disk or in another computer than when it is in the local memory of a computer. This access gap underlines the importance of efficient file buffering and prefetching. Knowledge of file usage patterns is a prerequisite for designing effective policies for file buffering and prefetching.

Processes are programs set in execution by users who initiated them, and processes use computer system resources such as processor time, memory, and file I/O. If resource usage requirements of a process can be predicted before it starts running, this knowledge can have applications in scheduling the process. For example, when multiple processors are available, as in a distributed computer system, predicted process resource requirements can be a sound basis for assigning the process to a lightly loaded processor. An additional motivation is in the area of reliable distributed computing: Knowledge of resource commitments can be useful in reorganizing a system under failure. This thesis is concerned with a measurement-based study of file access patterns and process resource usage prediction.

1.1. Goal of the Thesis

The goals of this thesis are: (1) to measure the logical-level file usage and process resource usage in a production system; (2) to develop and demonstrate practical approaches to file usage analysis that provide comprehensive knowledge of how files are used; and (3) to develop and demonstrate a resource usage prediction scheme.

File access and process scheduling are fundamental issues in a computer system. A measurement-based approach to understanding file and process resource usage in an existing system is

important for efficient implementation of these essential services in new systems. This research is timely because the present growth of computer technology is towards large and complex distributed systems, and these complex systems are expected to adapt to various workloads in order to provide efficient file access and an optimal load assignment.

1.2. Thesis Outline

Chapter 2 briefly reviews files and processes as implemented in the measured system. It describes the measured environment, and provides details of the measured data. This chapter also describes the measurement technique, and quantifies the effects of measurement on the system.

Chapter 3 introduces a user-oriented analysis of file usage, and demonstrates its application to file usage data collected from the production system. The key aspect of this analysis is a characterization of users and files using a few file usage measures. It is shown that this approach identifies differences in files as well as users.

Chapter 4 describes an analysis of file references that was done on file usage data collected from the same system when it was running a later version of the operating system. Unlike the user-oriented analysis, this analysis provides reference-level usage information and time-based measures about file usage, instead of characterizing either users or files. Results from this analysis is shown to complement those obtained from the user-oriented analysis.

Chapter 5 deals with process resource usage prediction. A statistical cluster analysis of processes is described. Resulting clusters are used to build a resource usage model for past executions of a program, and this model, in turn, is used to predict resource requirements of the program's next execution.

Chapter 6 summarizes the thesis and suggests future research directions for this work.

CHAPTER 2

PROCESSES, FILES, AND THEIR USAGE MEASUREMENT

In this chapter, we describe the data used in this study, the measurement technique employed, and the measured environment. Also, a brief review of processes, files, and operations provided for their manipulation in the Berkeley versions of UNIX is given, so that the terminology used here and in the rest of the thesis will be clear. For more details on UNIX-related issues the reader is referred to [Quarterman 85; Ritchie and Thompson 78; Berkeley UNIX 84; Berkeley UNIX 86].

2.1. Processes and Files in UNIX

Most user activity in UNIX is centered about processes and files. Several *system calls*, the procedural interfaces to UNIX kernel, are provided for their creation and manipulation.

2.1.1. Processes

In UNIX, a process is a program in execution [Quarterman 85]. To run a new program, a *fork* followed by an *execve* system call is used. A fork creates a new process with an almost identical data space. An execve replaces the virtual memory space of a process with that of a program by reading its executable module from a file. Processes are identified by their *process identifier* or *pid*, which is an integer returned by the fork system call. Just before terminating, every process executes an *_exit* (not to be confused with *exit*) to do house-keeping chores such as system accounting.

2.1.2. Files

In UNIX, data is kept in files as a sequence of bytes. A file must be opened with an *open* system call before its contents can be accessed. Open system call translates a symbolic file name into an *inode* number, checks access rights, and returns a *file descriptor*. A file descriptor is an index into an in-core file table maintained in the UNIX kernel. A *read* system call copies a specified number of bytes from a

file, starting from the current reference point, into a program defined data structure. A *write* system call copies a specified number of bytes from a program's data structure to a file, storing them at the current reference point. After a read or a write, the reference pointer is automatically updated to indicate the current reference point, but it can be changed with an *lseek* system call. A *close* system call closes the file. Besides these, several other system calls are provided to access and manipulate the contents of the inode and the file table entry. This study does not concern itself with these other system calls because either they are infrequent or they consume very little of system resources.

Directories are special purpose files that contain information on how to find other files. With the help of directories, files in UNIX are organized into a tree-structured hierarchy. Directories can be accessed directly using the above described system calls as well as few special ones, such as *mkdir* and *rmdir*. Indirectly, one or more directories are always referenced when translating a file name into its inode number (during an open call, for example).

2.2. Measured Environment

The data analyzed in this study was collected from a VAX-11/780, which was first running 4.2 BSD UNIX and later 4.3 BSD UNIX. The system had 8M bytes of main memory and over 300M bytes of secondary storage. It was used by the faculty and graduate students of the Department of Computer Science, University of Illinois at Urbana-Champaign, for text editing, sending and receiving mail, and for research programming. About 300 logins were recorded per day, but at any time the system only had a maximum of 40 users. System load average (a time-varying measure indicating the number of ready-to-run processes in the system) ranged from 0.05 to 10.0 during the measurement period.

2.3. Measured Data

The data for this study was collected in two separate measurements. Table 2.3.1 summarizes the information gathered. The first measurement took place during a period from March through May in 1985. File-related system calls and their arguments were recorded on a continuous trace basis. Since

Table 2.3.1: Summary of the Data Collected.

measurement	system call	information gathered
measurement #1, file-related system calls.	all	user id, process id, file id, and time.
	create, open	mode of open, file type, and file size.
	close	file size.
	read, write	bytes accessed, starting offset
	link, unlink	target
	trunc	truncation length
	mkdir, rmdir, stat, chdir	-
measurement #2, process-related system calls.	all	user id, process id, and time
	fork	child process id.
	execve	id of the program being executed.
	_exit	contents of *rusage* structure.
measurement #2, file-related system calls.	all	user id, process id, file id, and time.
	create, open	mode of open, file type, and file size.
	close	bytes read, written, and file size.
	lseek	old and new reference points.
	link, symlink, unlink, rdlink	target
	trunc	truncation length
	mkdir, rmdir, stat, chdir	-

the intent of this study was to analyze users' file usage that was not influenced by the buffering policy or by the level of multiprogramming in the system, system call level data rather than disk I/O level data was collected. Also, the data measured was limited to users' data files and to files belonging to the Notes[1] file system. Specifically, it did not include calls to the UNIX command files, the operating system related log, database, and library files. This was done by filtering out calls to files owned by the system identifiers such as *root* and *bin*. (References to the excluded files were traced in the next measurement.) In addition to file access data, user login information was also collected so that each system call could be related to a login session. The data was collected from 8:00 a.m. to 12:00 midnight on Monday through Friday, each day being selected from a different week. The hours capture the typical working hours of most users. The five days of data collection were randomly selected from five different weeks so that the data represents a good sample of system usage. A total of over 2000 logins and over 1.5 million system calls were recorded.

In the second measurement, file-related system calls (except read and write), as well as fork, execve, and _exit system calls were traced on a continuous basis, without excluding references to any files. Since references to all files were included, it was necessary to avoid tracing individual read and write calls, to keep measurement overhead low. Instead, the in-core file table structure of UNIX kernel was modified to record the number of bytes read and written for each opened file, and this information was collected at file close. Process-related system calls recorded information about process creation and resource usage. Resource usage information was taken from the *rusage* structure maintained by the UNIX kernel. In this measurement, the data was collected for one week starting from 8:00 a.m. on Tuesday, April 21, 1987. A total of over 65,000 processes and over 2.5 million file open-close combinations were recorded.

---

[1]Notes is a multi-topic bulletin-board-like system. Messages for a topic are stored together in one file; users can selectively read messages and can also add new messages. See [Essick 84] for more details. Similar bulletin-board systems are available on most computers, and in some installations, the system is known as News and has a slightly different implementation.

Data collected in both measurements, however, did not include references to directories that were made indirectly while translating a file name to an inode number. These indirect references were excluded because the mechanism used (buffering, for example) is quite different from the direct access and hence cannot be studied together.

2.4. Measurement Technique

The data was collected using a circular buffer in the kernel area. the UNIX kernel code was modified in such a way that when a process invoked a system call that was selected for tracing, it filled an entry in the buffer with proper information. A user-level process woke-up at regular intervals (e.g., 5 seconds) to read the buffer contents and to store the data on disk. Care was taken to avoid recording the activity of the measurement process itself. The measurement had little effect on the system: The circular buffer used less than 1% of memory available, the user-level process used less than 0.1% of CPU time, and users never complained.

In summary, the data in this study was collected from a university research environment. The data consisted of the logical file I/O, in the form of file open, close, and so on, and process resource usage such as CPU time, and memory usage. The measurement was carried out with minor modifications to the kernel that included a provision of a circular buffer, and had no adverse effects on the regular use of the system.

CHAPTER 3

A USER-ORIENTED ANALYSIS OF FILE USAGE

This chapter describes a user-oriented analysis of file usage based on data collected from a VAX-11/780 running 4.2 BSD UNIX. The measured data is a trace of file-related system calls (read, write, open, close and other calls with their arguments), and is described as the first measurement data in Chapter 2. The data is analyzed to characterize users and files.

3.1. Overview

This analysis quantifies a typical user's file usage in a login session and the usage of a typical file in all login sessions, which is a departure from the traditional approach of analyzing file references. A measure of file usage referred to as *accesses-per-byte* is introduced. This measure combines fraction referenced and number of references to a file. Using this measure, two types of usage characterizations are defined. A typical user's file referencing behavior is quantified by the average accesses-per-byte made to referenced files in a login session, the average size of referenced files, and the number of files referenced. This characterization is referred to as a *user characterization*. The usage of a typical file is quantified by the average of accesses-per-byte made over all login sessions, the average file size, and the number of login sessions that referenced the file. This characterization is referred to as a *file characterization*.

Files are then categorized according to the UNIX file type (regular or directory files), the ownership, and the type of use (read-only, temporary, etc.); users are categorized by the amount of file I/O during a login session. Based on empirical distributions and on analysis of variance, the user and file characterizations are shown to quantify the variability in file usage across the file and user categories. Thus, we establish a systematic approach to quantify a user's file usage in detail, and show that the analysis distinguishes nonuniformity in file usage.

The other results from the study are the following. Almost all user-owned files are completely referenced. User-owned files are usually small and are not referenced many times in a login session, but heavy users' files are larger and are referenced several times more than those belonging to light users. Even though overall file sharing is small, some files belonging to the bulletin board (Notes) system were accessed by many users (simultaneously and otherwise). A surprisingly large number of users (over 50%) are found to reference files belonging to other users; some group programming efforts and system utilities (such as *finger*) were the reasons for this result.

The organization of the remainder of this chapter is as follows: Section 3.2 discusses the related work in this area. Section 3.3 recaps the description of the measured data. Sections 3.4 through 3.7 discuss the user and file characterizations in detail. In section 3.8, we briefly speculate on how the results might be used in file system design. Summary and conclusions appear in section 3.9.

3.2. Related Work

Related work can be categorized as the long term and short term file usage studies. The long term studies analyze data from once-a-day scans of the file system. The scans of the file system record whether or not a file is referenced on a day. Consequently, the studies such as [Smith 81] and [Satyanarayanan 81] do not quantify how heavily a file is used during a day. A comprehensive review of long-term studies can be found in [Satyanarayanan 81].

The short term studies analyze traces of disk I/O requests or system calls. Based on traces of disk I/O requests from two IBM batch systems, in [Porcar 82], an approach for shared file migration assuming a Markov chain model for the file usage is described. In the model, each state corresponds to a node in a computer network. In calculating model parameters, aggregate referencing behavior of all users is used. As the analysis in this chapter will demonstrate, such an assumption is not valid in general. Some users can vary significantly from the norm in their referencing characteristics. Consequently, model parameters can also vary for these users, and thus affect the validity of the overall model in a dynamic sense. Since no attempt was made to validate the Markov model itself, the impact

of user variability on the results is unknown. Another study of short term file access [Ousterhout 85], mainly analyzes disk cache performance.

The study closely related to the present one is that in [Floyd 86a] and [Floyd 86b]. Using short term file access data from a 4.2 BSD UNIX environment, the author provides distributions of measures such as fraction referenced, file-open time, inter-open time, and number of references per file. This broad analysis of references to all types of files, also brings out the value of a short term file usage study. As the author points out, an important issue, which may enhance the value of this work, is an in-depth analysis of file usage activity by user.

None of the short term studies explicitly quantify a typical user's file usage. As will be shown, user-based and file-based measures quantified in this chapter are useful in bringing out differences in users (and in files), and these differences can be important in evaluating an existing system. The work presented here is unique in the following respects:

- The notion of *how heavily a file is used* is quantified.
- A typical user's file usage as well as usage of a typical file by all users are quantified.
- The above two ways of characterizing file usage are shown to distinguish nonuniformity in file usage.
- Properties specific to file categories (e.g. user-owned, notes files, and others) and user categories (light and heavy) are evaluated.
- Analysis of variance methods are used to evaluate the relative influence of the user and file categories on usage characterization measures.

3.3. Data Description

File-related system calls and their arguments were traced on a continuous basis, from a VAX-11/780 running 4.2 BSD UNIX (as described more fully in Chapter 2). For each file and login session combination, the following data was obtained from the trace and is used in the analysis done here.

User identification data:

- user id
- login process id

File specific data:

- file id (inode, device, and *usage* numbers)
- file size
- file owner's id
- file type information

File usage data:

- number of reads
- average bytes read in each read call
- number of writes
- average bytes written in each write call

Time stamps:

- software clock value at the first and last call

The data analyzed is limited to users' data files and to files belonging to the notes file system. Specifically, it was decided not to include calls to command files and system files (operating system related log, database, and library files) in this analysis. The exclusion was achieved by filtering out calls to files owned by the system identifiers *root* and *bin*. The reasons for the exclusion are detailed below.

Command files are the load modules containing executable programs. Once execution of one of these files begins, the virtual memory system is responsible for making pages of the program available in memory. Paging behavior of programs has been extensively studied elsewhere, and it is not our objective to duplicate this work.

Here, we are primarily concerned with the analysis of users' files. The usage patterns of the system files can be substantially different from that of users' files--system files are usually referenced only in part, although (sometimes) heavily. An example is the file that contains users' passwords and other related information, */etc/passwd*. As it will become apparent in the subsequent sections, users tend to access their own files in entirety. Thus, the inclusion of the system files in our analysis can significantly distort the overall results.

Further, the referencing patterns of the system files can depend on the specific implementation of the operating system. For example, in Version 4.3 of the Berkeley UNIX the password file is searched by hashing, whereas a sequential search is employed in Version 4.2. In SUN Microsystems UNIX, most system databases are implemented using centralized server processes. Given that the referencing patterns of system files are different from user files and that the referencing patterns of the system files can change from one implementation of UNIX to another, we believe that the system files should be studied separately.

The user files, by their very nature, are independent of implementation. Therefore, the analysis of the user files can be of considerable value in creating a synthetic workload that is useful for any system. It should be emphasized that the key issue in this study is methodology, and the method is equally applicable to the analysis of the system files.

In summary, the data used in this study is traced from a university research environment, and consists of file-related system calls to system-independent files,[1] namely the users' data files and notes files.

3.4. File Usage Characterization

In this section, we introduce two types of characterizations of file usage. A *user characterization* quantifies how a user uses an average (referenced) file in a login session, and a *file characterization* quantifies how a file is used by an average user in the measurement period. Alone, neither the user characterization nor the file characterization fully captures the many-to-many relationship between users and files. For instance, the user characterization does not show file sharing among users, but the file-based approach does. On the other hand, the file characterization does not show variability in users, which the user-based approach quantifies. In addition, as will be shown later, the two ways of characterizing the same data allow us to quantify the nonuniformity in file access.

---

[1]Indirect references to directories for file name translation are also excluded. The argument for the exclusion is similar to the one given for the system files. This indirect use of directories is quite different from the normal usage, and the implementation can change from one system to another. Consequently, these indirect references should be studied separately, as is done in [Floyd 86b].

A key measure central to both characterizations is what will be referred to as the number of *accesses-per-byte (APB)*. Given a login session s and a file f, the APB for the specified file in the login session is defined as:

$$Accesses\_Per\_Byte[s,f] = \sum_{i=1}^{NumOpens} FR[s,f,i] \qquad \text{Eq. 3.4.1}$$

where, $FR[s,f,i]$ is the fraction of the file referenced in *ith* open of the file, and *NumOpens* is the number of opens made to the file in the login session. Intuitively, the measure shows how many times a file is completely referenced by a user in a login session, and thus quantifies *how heavily a file is referenced.* As it will be seen, this measure allows us to clearly classify who are heavy users in the system.

If the fraction referenced, for a given file, is always 1.0, then the APB shows number of references made to the file. However, if only one reference is made to the file in a login session, then the APB, in common with other file access studies [Porcar 82; Floyd 86a], measures the fraction referenced. But unlike these studies, accesses-per-byte (as it combines fraction referenced and number of references) also provides information on how heavily a file is used in a given period of time. Our data shows that in nearly 92% of references, the referenced file is accessed in its entirety. For files not referenced in entirety but referenced many times, such as operating system related log and database files, the APB (in Eq. 3.4.1) should be calculated for each record of the database.

We considered alternatives to the accesses-per-byte measure, such as accesses per logged-in minute and accesses per day, but found them not to reflect a user's file usage characteristics. For example, accesses per logged-in minute may depend on the system load. If a user's login session occurred when the system load is high, then the user's accesses per minute measure can be significantly lower than what it would be if the user were logged-in at a low system load. Thus, accesses per minute may be more reflective of the system usage than a user's file usage. Another point of importance in this regard is that, as will be shown later in the chapter, if a user's total file I/O in a login is high then (a) the user's file I/O rate is also *likely* to be high, and (b) the user's accesses-per-byte is also *likely* to be high.

Consequently, if a user's APB is high it is *likely* that the user's file I/O rate is also high. So, since the accesses-per-byte measure reflects file I/O rate to a large extent without actually being influenced by the system load, we chose to use it as the characterization measure of a user's file usage.

The other alternative, accesses-per-day, may encompass too much of a user's activity, and thus it may suppress the variability in usage. For example, a user may login several times during a day, doing different things in each login, and these differences will be averaged out in accesses-per-day.

One can ask: Why analyze file usage by user and by login session? Most current literature does not do so. For example, the study in [Porcar 82] assumes that all users are alike. As we will show, the distributions of file usage measures can be heavily skewed by a few, but significant number of heavy users. In such a case it is invalid to assume uniformity among users. In fact, in analyzing user behavior, we found that users can indeed behave differently in different login sessions. Thus, it was considered statistically sound to treat each login session separately. Finally, one application of this analysis, synthetic workload creation, needs user-based as well as file-based analysis.

Based on the accesses-per-byte measure and a few other parameters, we define the user and the file characterization measures.

User Characterization: Each user is characterized by the average number of accesses-per-byte made to referenced files, the average size of the referenced files, and the number of files referenced in a login session. Mathematical definitions[2] for the characterization measures of *ith* user with N_i files follows:

$$accesses\_per\_byte[i,*] = \frac{1}{N_i}\sum_{j=1}^{N_i} accesses\_per\_byte[i,j]$$

$$file\_size[i,*] = \frac{1}{N_i}\sum_{j=1}^{N_i} file\_size[i,j]$$

$$num\_of\_files[i,*] = N_i$$

---

[2]*Notation:* In the mathematical expressions, *accesses_per_byte[i,j]* denotes accesses per byte made to jth referenced file by ith user. A "*" in the place of an index indicates a quantity obtained by averaging over the index. Similar notation is employed for other measures

File Characterization: Each file is characterized by the average number of accesses-per-byte made by all logins in the measurement period, its average size, and the number of users of the file. Mathematical definitions for the characterization measures of *jth* file with M_j users follows:

$$accesses\_per\_byte[*, j] = \frac{1}{M_j}\sum_{i=1}^{M_j} accesses\_per\_byte[i, j]$$

$$file\_size[*, j] = \frac{1}{M_j}\sum_{i=1}^{M_j} file\_size[i, j]$$

$$num\_of\_users[*, j] = M_j$$

3.4.1. Distributions of the Characterization Measures

In this subsection, distributions of the user and file characterization are provided, with intuitive explanations for the results. Statistical models to fit the distributions are also provided. Figures 3.4.1 and 3.4.2 show the distributions and the multi-stage gamma functions (g's in the figures) model the distributions. Mean and quartiles of the distributions appear in Table 3.4.1 and Table 3.4.2, where the parenthesized values are the standard deviations of the parameters across the five days of measurement. Representativeness of data is evident from small standard deviations.

As seen in Figure 3.4.1, distributions of the user-based measures are skewed towards small values, and they also have long tails. This is also evident from the fact that mean values are larger than their median values but are smaller than third quartiles. It implies that even though there are many light users, a significant number of heavy users also exist. Since these heavy users make severe demands on the system, all users can experience poor response times when a heavy user is active (assuming shared resources). From a file system designer's viewpoint it is important to differentiate these heavy users so that the file system can be designed to adapt to different workloads. From a performance evaluator's viewpoint, such a characterization helps to accurately evaluate the system performance under heavy and light loads.

The user-based file size distribution (Figure 3.4.1) shows two peaks, the second peak occurs near 14K bytes. However, the other measures show little difference between the users with mean file size

Table 3.4.1 : Means and Quartiles of the User Characterization Measures

measure	mean	median	III quartile
accesses-per-byte	1.57 (0.06)	1.34 (0.04)	1.78 (0.11)
file size	14.57k (1.318)	9.75k (0.433)	24.12k (2.96)
number of files	27.94 (2.09)	15.60 (1.14)	33.55 (3.34)

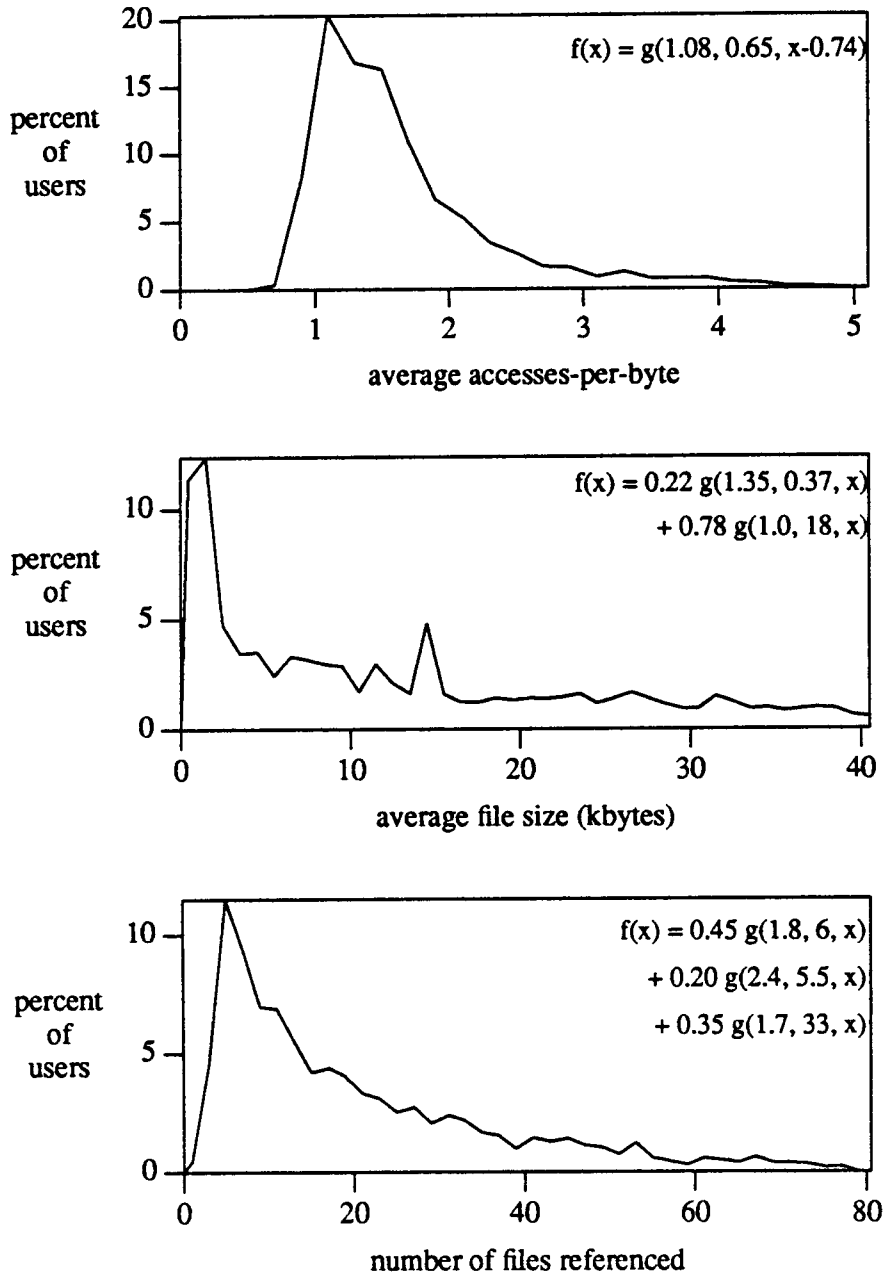


Figure 3.4.1: Distributions of the User Characterization Measures

greater than 14K and those with mean file size less than 14K. A further examination reveals that the users belonging to the former group referenced mostly notes files, which are considerably larger than the other files. This group accounts for about 45% of the total users.

Distributions of the file-based measures (Figure 3.4.2) have even longer tails than distributions of the user-based measures. For instance, the mean of the file-based accesses-per-byte is larger than its 3rd quartile. The file-based file size distribution (Figure 3.4.2) shows dominance of small files in a UNIX environment. About 80% of all files are smaller than 10K bytes. Studies of long term file reference patterns (for example, in [Smith 81] and [Satyanarayanan 81]), reported similar file size distributions.

Owing to the long tails and multiple modes, the empirical distributions are modeled by multi-stage gamma distributions. The probability density functions appear in figures 3.4.1 and 3.4.2 as:

$$f(x) = \sum_{i=1}^{N} w_i \, g(\alpha_i, \theta_i, x - s_i)$$

where w_i is the weight, and s_i is the offset of the *ith* stage. N is the number of stages. Sum of all w_i is 1. G is the gamma distribution [Hogg and Tanis 83] function:

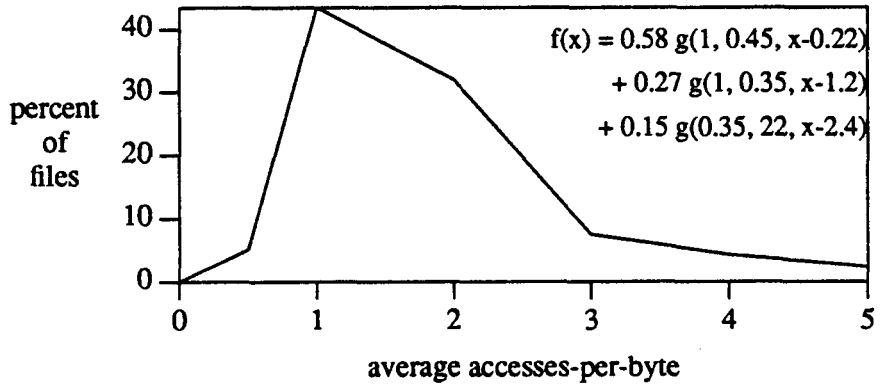
$$g(\alpha, \theta, y) = \frac{1}{\Gamma(\alpha)\theta^{\alpha}} y^{\alpha-1} e^{\frac{-y}{\theta}} \quad 0 \le y < \infty$$

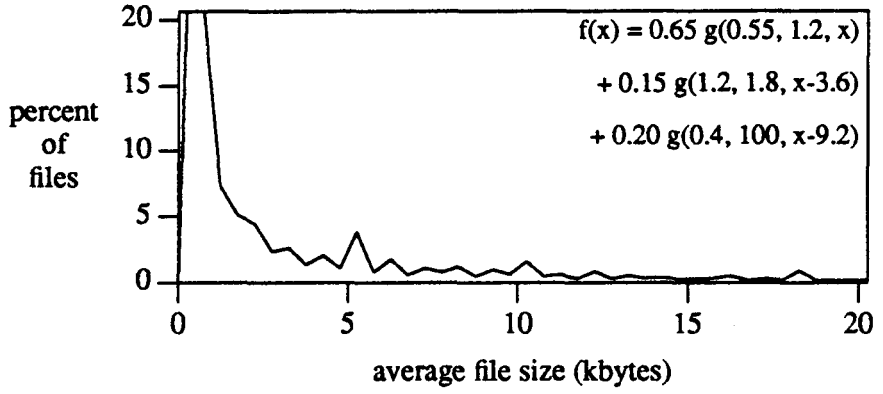
The Kolmogorov-Smirnov test [Daniel 78] shows that the multi-stage gamma distribution models the empirical distributions at over 99% confidence level. We could not fit multi-stage exponential models to the same degree of accuracy. Clearly, single stage exponentials are not valid representations of the measures. Most analytical performance evaluation studies of file systems assume workload parameters have exponential distributions because the system models then become numerically tractable. However, our results question the validity of such exponential assumptions.

In summary, distributions of the user and file characterization measures follow a multi-stage gamma distribution. Hence, single stage exponential models appear to be invalid for these measures -- a result of significance in performance evaluation. Also, there are some heavy users and large files that significantly effect the distributions, which clearly demonstrates that using aggregates is not

Table 3.4.2: Means and Quartiles of the File Characterization Measures

measure	mean	median	III quartile
accesses-per-byte	2.35 (0.09)	1.66 (0.12)	2.00 (0.00)
file size	11.38k (1.54)	1.42k (0.22)	7.03k (0.76)
number of users	2.00 (0.11)	1.00 (0.00)	1.4 (0.55)





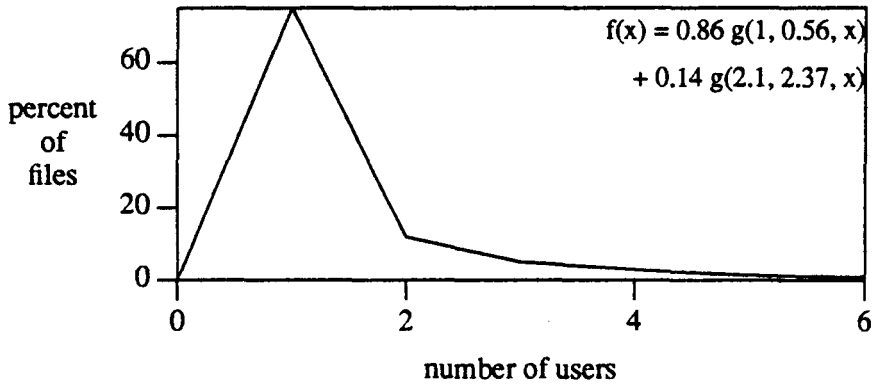


Figure 3.4.2: Distributions of the File Characterization Measures

satisfactory. In an attempt to further quantify the differences in users and files, the next two sections explore various categories of files and users.

3.5. Effects of File Categorization

So far we have obtained distributions of the user and file characterizations. How these characterizations change with different *file categories* is brought out in this section. In particular, we examine how a user uses files belonging to different categories, and how a file belonging to a given category is used in all login sessions. Further, a comparison of the corresponding measures of the user and file characterizations shows nonuniformity in file access. For the purposes of this study, files are categorized using the following orthogonal criteria:

1. **UNIX file type**: A file may be a directory (DIR) or a regular file (REG). This criterion groups the files according to the implicit use of the files in the operating system.

2. **Ownership**: A file of the notes file system belongs to NOTES type, a user-owned and owner-referenced file belongs to USER type, and a user-owned nonowner-referenced file belongs to OTHER type.

3. **Type of Use**: A file whose contents are only read during a login session belongs to RDONLY class. A file that is either nonexistent before or truncated to zero size before writing belongs to NEW class. A file that is nonexistent before and deleted after use is a temporary (TEMP) file. A file that is neither RDONLY nor NEW nor TEMP belongs to RD_WRT class.

A file category[3] is defined as a specific combination of UNIX file type, ownership, and type of use. For example, REG-USER-RDONLY refers to user-owned regular files that are used in a read-only mode in a login session. If the context is clear, a shorter name (e.g., while discussing regular files, REG-USER-RDONLY may be abbreviated as USER-RDONLY) is used to reference a file category.

3.5.1. User Characterization by File Category

This section discusses how a user uses files belonging to different categories, and the next section discusses how a file belonging to a given category is used in all logins. Table 3.5.1 shows the mean

[3]Note that how a user uses a file is the basis for the ownership and type of use classifications. Consequently, a file can be in more than one class. An examination of the data shows that about 5% of the files belong to more than one category. In developing file characterization, we consider such multiple occurrences of a file as occurrences of multiple files.

values of the user characterization measures by file category. (Figure A.1 shows distributions of the user characterization measures for selected file categories.) For example, an average user's usage of a REG-USER-RD_WRT file is characterized by 3.46 accesses-per-byte and 19796 bytes of file size. On an average, 2.1 REG-USER-RD_WRT files are referenced in a login session. About 45% of logins reference files of this category.

An average user's usage of REG-USER files: An average read-write file is about ten times larger than an average read-only file, and is accessed 3 times as much. This is because, in UNIX, read-only files contain mostly default options, electronic mail messages, and user defined type declarations. Therefore, the read-only files are usually small and are rarely modified. On the other hand, read-write files contain program source code, object modules, or text. As a result, they are relatively large and are frequently updated. These statistics indicate that migration or prefetching an entire file may be a more efficient strategy for all REG-USER files. Specifically for read-write files, a delayed write-back policy is worth considering, because these files are heavily used in a login session. However, reliability requirements may dictate regular write-backs to nonvolatile storage (disk), but during heavy usage periods, these

Table 3.5.1: Averages of the User Characterization Measures by File Category

file category			characterizing measures			%users using the category
file type	owner	type of use	accesses-per-byte	file size	files	
DIR	USER	RDONLY	3.33	803	2.8	68%
	NOTES	RDONLY	2.41	6248	1.0	8%
	OTHER	RDONLY	2.28	1198	2.5	70%
REG	USER	RDONLY	1.38	1909	5.8	100%
		NEW	2.30	11323	4.0	40%
		RD_WRT	3.46	19796	2.1	45%
		TEMP	2.00	9233	9.7	60%
	NOTES	RDONLY	0.54	49856	10.1	53%
		RD_WRT	1.77	20254	5.7	38%
	OTHER	RDONLY	1.52	4280	3.0	51%

write-backs can cause response time degradation [Johnson 87]. Thus, it is preferable to improve memory reliability instead of frequent write-backs [Georgiou 87].

An average user's usage of REG-NOTES files: Read-only and read-write files are the largest and the next largest (49856 and 20254 bytes). On an average, only 54% of a NOTES file is read in a login session. Even read-write files are not fully accessed (accesses-per-byte is 1.77). In contrast to the above, migration or a complete prefetch of these files is inadvisable as it would waste file buffer space as well as communication bandwidth. Thus, different policies are suggested for different file categories.[4]

An average user's usage of directories: As expected, an average USER or OTHER directory referenced in an average login session is only about 1K bytes. A user accesses directories two to three times as heavily as REG-RDONLY files, but the number of directories referenced is only half as many as regular files. This indicates that even a small per-user directory-cache can achieve very high hit ratios, and is worth investigating.

Probability that an average user references a file category: The last column in Table 3.5.1 gives the probability that a user references a file of a certain category.[5] For example, the probability that a user reads one or more NOTES files is 0.53. Note that the categories are not mutually exclusive.

An average user's usage of other users' files: The last column of Table 3.5.1 also shows that there is a measurable degree of sharing[6] apart from NOTES files. Seventy percent of logins read directories and 51% read regular files that belong to other users. This unexpectedly large amount of sharing comes from two sources: first, there are a few research groups developing large software systems (e.g. a programming environment), and individuals involved in such projects share type-declaration files;

---

[4]Current implementations of UNIX use a single policy for all files.

[5] The last column of Table 3.5.1 shows that only 69% of users (i.e., 31% of users do not) read their own directories. At first it might seem improbable, but note that about 32% of users make file I/O less than 10K bytes (see section 6), and that our analysis does not include directory references made while translating a file name into an inode number.

secondly, UNIX provides utilities (e.g. *finger*) which enable a user to obtain information about another user by reading this other user's file (e.g. *.plan*). Interestingly enough, an average user accesses other users' files just as heavily as his own read-only files.

3.5.2. File Characterization by File Category

This subsection discusses how a file belonging to a given file category is used in all login sessions. Table 3.5.2 shows mean values of file characterization measures by file category. (Figure A.2 shows distributions of the file characterization measures for selected file categories.) For example, an average REG-USER-RD_WRT file is characterized by 4.30 accesses-per-byte, and 17443 bytes of file size. On an average, a REG-USER-RD_WRT file is referenced in 1.4 logins. Files of this category constitute about 4.7% of all files.

The last column of Table 3.5.2 gives the breakdown of files into file categories. About 75% of files are regular files that are user-owned and -referenced, and an additional 7% are directories of the same category. A little less than 10% of files are NOTES files. Over 4.6% of files are nonowner-

Table 3.5.2: Averages of the File Characterization Measures by File Category

file category			characterizing measures			%files in the category
file type	owner	type of use	accesses-per-byte	file size	logins	
DIR	USER	RDONLY	3.55	713	1.70	7.8%
	OTHER	RDONLY	2.21	708	3.43	3.4%
REG	USER	RDONLY	1.81	4524	1.83	21.5%
		NEW	2.54	11164	1.08	9.8%
		RD_WRT	4.30	17443	1.40	4.7%
		TEMP	2.00	12393	1.00	38.7%
	NOTES	RDONLY	0.80	31514	5.54	6.5%
		RD_WRT	2.68	19410	4.53	3.3%
	OTHER	RDONLY	2.36	8639	2.14	4.6%

[6] Does not necessarily imply simultaneous use.

referenced user files. These percentages show that, although most files are exclusively referenced by their respective owners, a significant portion (nearly 15%) of files are shared. Dominance of read-only files is also apparent: About 72% of all the permanent files are referenced in a read-only mode.

Accesses-per-byte and file size appear in Table 3.5.2 as well as in Table 3.5.1, and the corresponding entries in both tables exhibit certain similarities. This issue will be further discussed in the next subsection. Here, the key issue is file sharing, we comment on three types of sharing among users.

Sharing via notes files: From the logins measure of Table 3.5.2, it can be seen that an average NOTES file is read in 5.54 login sessions. Considering that nearly 150 different users use the system every day (at a rate of about 2.7 logins per person), one would expect a typical NOTES file to be used in more logins than this. A visual examination of the data reveals the presence of several special purpose NOTES files (such as a NOTES file exclusively used by a small research group) that influenced the characterization.

Simultaneous sharing via notes files: A separate analysis of notes file usage for a single day showed that over 2% of notes files are shared simultaneously by two or more users. One file had 4 simultaneous users at one time, and another file had 2 simultaneous users on 16 occasions during a day. Note that 22% of notes files had 3 or more (not necessarily simultaneous) users during the day, and nearly 10% of these notes files had 2 or more simultaneous users. These results indicate that a few notes are heavily shared.

In the previous subsection, it was observed that a typical user does not access notes files heavily, but here we showed that a few notes files are extensively shared (simultaneous and otherwise). These results may have some implications when considering a distributed environment. For example, the results, when applied to such an environment, suggest that the notes files (instead of being duplicated or buffered at each node) should probably be supported using centralized *servers* similar to what is done with the password files in SUN Microsystems UNIX.

Sharing via users' files: Table 3.5.2 also shows that an OTHER class (nonowner-referenced user class) file has 2.14 users. This result complements a related observation from the previous subsection, which indicates that an average login session references 3.0 files of the OTHER class. Thus, between the two, the user and file characterizations well quantify the degree of file sharing.

As the results indicate, in a single processor system, users do take advantage of the ability to access other users' files, which shows the value of integrating single-user workstations into a unified system. However, since the usage of the OTHER class of files is less frequent than the rest of the file categories, performance optimization for the OTHER files may not be a real concern. Thus, a simple scheme such as SUN NFS may be adequate, and extensive migration policies may be unnecessary in these situations.

3.5.3. Comparison of the User and File Characterizations

Since the user characterization describes a typical user's usage of an average file, and the file characterization describes the usage of a typical file by an average user, the extent to which these characterizations are similar shows the uniformity in file usage. This point is brought out when tables 3.5.1 and 3.5.2 are compared with each other. Even though both tables display a similar trend, significant differences can be observed. The file characterization measures are reflective of heavy users, and the user characterization measures are typical of light users. For instance, accesses-per-byte measure in Table 3.5.2 (i.e., in the file characterization) is larger than in Table 3.5.1 (i.e., in the user characterization). In particular, the difference is about 35% for REG-USER files, and it is over 50% for read-write notes files. The reason for these results is that a heavy user tends to reference a large number of files, and consequently his activity influences the file characterization considerably. On the other hand, a majority of logins in the measured system are light, and consequently the user characterization reflects their behavior.

File sizes of REG-USER files also follow the pattern of the accesses-per-byte measure, but the NOTES files are an exception. For example, file size of a read-only NOTES file is about 50K bytes in

Table 3.5.1, whereas in Table 3.5.2 it is only about 30K bytes. An explanation is that a few large NOTES files are read by many users, but since these files constitute only a small percentage of all NOTES files they do not influence the file characterization much. However, it implies that high throughput as well as fragmentation avoidance is needed for large files.

The next section introduces a user categorization, and discusses how the user categorization explains the nonuniformity in file access.

3.6. Effects of User Categorization

Based on logical file I/O done, we categorize users as casual, light, medium, heavy, and very-heavy. The logical file I/O of a user is the total number of bytes read from or written via the read and write system calls in a login session. Mathematically, it is:

$$File\_IO = ReadCalls * AvgReadSize + WriteCalls * AvgWriteSize$$

Table 3.6.1 shows the percentage of users in each user category. Note that the system usage is fairly heavy: Over 42% of users have done file I/O in excess of 100K bytes per login session.

Tables 6.2 and 6.3 show the user and file characterizations by user category. For the sake of brevity, the measures are shown only for the USER, NOTES, RDONLY, and RD_WRT file classes. Figure B.1 shows distributions of the user-based measures for user-owned files and for heavy and light users.

A significant result from Table 3.6.2 is that the user characterization measures (i.e., APB, file size, and number of users) follow file I/O done by the user. For instance, a very-heavy user's usage of

Table 3.6.1: User Categories by File I/O

user category	file I/O range	percent of users
casual	less than 1K bytes	8.7%
light	1K - 10K	23.5%
medium	10K - 100K	25.1%
heavy	100K - 1,000K	33.8%
very-heavy	1,000K or more	8.9%

Table 3.6.2: Averages of the User Characterization Measures by User Category

measure	user category	values by file category			
		USER		NOTES	
		RDONLY	RD_WRT	RDONLY	RD_WRT
accesses-per-byte	casual	1.01	-	0.03	-
	light	1.06	1.67	0.29	-
	medium	1.22	2.12	0.55	1.26
	heavy	1.45	3.46	0.61	1.93
	v-heavy	2.46	6.06	0.75	2.03
file size	casual	158	-	24271	-
	light	354	10505	23743	-
	medium	1558	12064	46580	21554
	heavy	2829	18794	58419	19607
	v-heavy	5266	41777	62761	23320
number of files	casual	2.30	-	1.00	-
	light	3.32	1.06	1.4	-
	medium	4.93	1.90	3.50	2.23
	heavy	7.32	1.88	13.4	6.01
	v-heavy	12.33	3.52	23.9	10.34

Table 3.6.3: Averages of the File Characterization Measures by User Category

measure	user category	values by file category			
		USER		NOTES	
		RDONLY	RD_WRT	RDONLY	RD_WRT
accesses-per-byte	casual	1.02	-	-	-
	light	1.06	1.52	0.60	-
	medium	1.24	2.29	0.64	1.50
	heavy	1.53	3.43	0.75	2.58
	v-heavy	3.10	8.20	0.82	2.80
file size	casual	153	-	-	-
	light	357	8316	18217	-
	medium	1875	13650	47157	23323
	heavy	4086	16218	31511	16269
	v-heavy	7133	28994	42213	21155
number of users	casual	1.58	-	-	-
	light	1.43	1.29	2.18	-
	medium	1.42	1.22	2.18	1.92
	heavy	1.47	1.27	4.49	3.66
	v-heavy	1.25	1.21	2.50	2.26

USER-RDONLY files is three to twelve times larger than that of a light user.[7] So, we conclude that the heavy usage can be quantified using any of the following measures: total file I/O, average accesses-per-byte, mean file size, or the number of files.

The blank entries in Table 3.6.2 are owing to the absence of certain file categories in the referenced files of a user category. For example, a casual user does not reference any read-write files. This information is part of a casual user's characterization. Turning now to Table 3.6.3 (the file characterization), it can be seen that accesses-per-byte and file size measures follow the same trend as in Table 3.6.2 (the user characterization).

Interestingly, a comparison of tables 3.6.2 and 3.6.3 shows smaller differences in the user and file characterization measures than in section 5.3, where user categories were not used. For example, differences in accesses-per-byte of REG_USER files are now about 8% compared to over 35% differences noticed in section 5.3. Similarly, differences in file sizes of REG-USER-RDONLY files are now about 35% compared to 120% earlier.[8] This closeness between the user and file characterization shows uniformity in file access among users of a user category. Recall that in section 5.3, the differences between the user and file characterizations were attributed to the nonuniformity in file access, and it was claimed that the user categories would reduce the nonuniformity. By making the users more uniform in each category, we have reduced the nonuniformity in each user category, thus, providing support to the claim made. These patterns are also apparent in Figure B.2, which shows distributions of the file characterization measures for user-owned files and for heavy and light users.

3.6.1. Correlation Between a User's Total File I/O and I/O Rate

Earlier in this section, the total file I/O done by a user was used to group users into heavy and light users. One could argue that a user's file I/O rate may be more significant than the total file I/O. Here, we show the correlation between a user's average file I/O rate and total file I/O. In Figure 3.6.3, each

[7]The only exception to this pattern is that a heavy user's NOTES-RD_WRT files are smaller than a medium user's files of the same category. This exception is partly responsible for the secondary peak in the file size distribution of Figure 3.4.1.

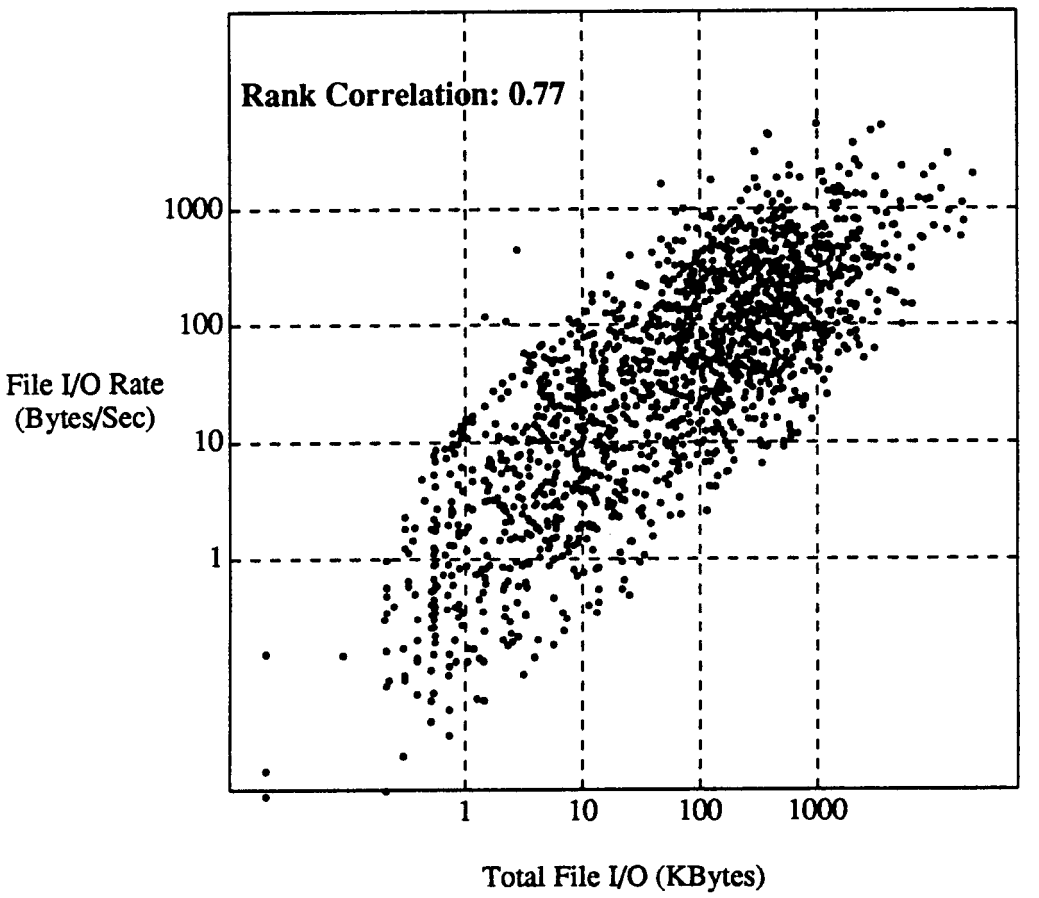


Figure 3.6.3: Users' Access Rate versus Total File I/O

user is denoted by a dot based on the user's file I/O rate and total file I/O done in a login session. A user's file I/O rate (bytes per second) is the average number of bytes read or written in a unit of login time. As shown in the figure, the coefficient of (Spearman's) rank correlation [Mendenhall and Sincich 84] for the two measures is 0.77. The rank correlation quantifies the relationship between the ranks of two quantities, and it shows how well high values of one measure correspond to high values of the other, without assuming a linear relationship between the two. A coefficient value of 1.0 implies a perfect correlation. Given that a coefficient value of 0.77 was observed, we can conclude that it is unlikely that a user categorization based on file I/O rate would be considerably different from the one based on total file I/O.

[6]Once again, an exception to this pattern is the size of the NOTES files.

In summary, an average user's characterization measures (average accesses-per-byte, average file size, and number of files) follow the total file I/O done by the user. Also, the user and file characterizations of a user category are almost identical, differences are as small as 8%. Applications of these results to file system design and evaluation will be (briefly) discussed in section 8.

3.7. The Relative Influence of the File and User Categorizations

In the last two sections, differences in the user and file characterization measures across file and user categories were quantified. In this section, we address two important questions:

- Are these differences statistically significant?
- What is the relative influence of many categorizations on the file usage measures?

We employ the *analysis of variance* (ANOVA) [Box 78] for this purpose. This is a well known statistical method for the quantification of the effects of several *factors* (e.g., file categorization criteria) on a *response variable* (e.g., accesses-per-byte). A linear dependency between the response variable and the factors is assumed, as in the following example:

$$Y = A + B + C + A\&B + A\&C$$

where A, B, and C are the factors and T is the response variable. A&B and A&C represent the interaction effects of A combined with B and C respectively. ANOVA decomposes the sum of square variations in Y (denoted by SST) into sum of square components of the terms on the right hand side of the model equation (SSA, SSB, and so on), and a residual error (SSE). The ratios, SSA/SST, SSB/SST, ..., and SSAC/SST, show the relative influence of the terms. The fraction SSE/SST represents unknown variations in the dependent variable. From the sum of square components, *significance levels* for the model and for each factor of the model are derived. The smaller the significance levels, the better the fit. For each measure, using mean values, an ANOVA model was obtained at better than 0.0001 level of significance. The model was analyzed using SAS, the Statistical Analysis System [SAS 85a; SAS 85b].

Table 3.7.1: ANOVA models for the file characterization measures and percent sum of squares contributions from the factors

source of variations (factors)	model for accesses-per-byte	model for file size	model for users
file_type	3%	23%	7%
ownership	19%	11%	50%
type_of_use	19%	-	4%
user_type	17%	-	11%
file_type&ownership	2%	14%	8%
ownership&type_of_use	-	36%	1%
user_type&file_type	-	16%	-
user_type&ownership	21%	-	-
user_type&type_of_use	13%	-	-
user_type&file_type&ownership	6%	-	18%
R-Square	0.78	0.74	0.89

Each column in Table 3.7.1 shows an ANOVA model for a characterization measure -- a nonblank entry implies the presence of the corresponding categorization (or an interaction of categorizations) in the measure's ANOVA model. For instance, an ANOVA model for accesses-per-byte is shown below:

$$\begin{aligned} accesses\_per\_byte = {} & file\_type + ownership + type\_of\_use + user\_type \\ & + file\_type\&ownership + user\_type\&ownership \\ & + user\_type\&type\_of\_use + user\_type\&file\_type\&ownership \end{aligned}$$

The relative influence of the categorizations are shown as percent sum of squares explained by each categorization (or an interaction of categorizations). A large percentage implies a heavy influence. As the results indicate, the variations in the characterization measures are statistically significant.

We find that the user type has the largest influence on accesses-per-byte. Alone, user type contributes 17% to variations in accesses-per-byte, and interaction terms involving user type contribute an additional 40% to variations in accesses-per-byte. Ownership of a file and type of use also figure significantly in explaining the variations in accesses-per-byte.

File type and ownership determine the file size. File type and ownership contribute 48% to variations in file size, and the interaction terms involving file type or ownership with other

categorizations contribute the remaining 52%.

The number of users of a file is mostly determined by its ownership. Ownership alone contributes about 50% to variations in the number of users, and the interaction terms involving ownership contribute an additional 27%.

(The effects of the categorizations on user characterization measures were also analyzed for statistical significance and relative influence. The results are shown in Table C.1 of Appendix C.)

3.8. Implications of the Results

Throughout this chapter we have obtained numerous results on both user and file characteristics, and discussed specific implications of these results. This section highlights important results and discusses possible implications for efficient file system design and evaluation.

A. Synthetic Workloads for File System Evaluation

The measures and distributions from this study can be used to develop a synthetic file access workload for evaluating the file system of a stand-alone or a networked system. Such a workload generator has been developed, and is described in [Barrington 86]. Briefly, the workload generator first populates disk(s) with files using the file size distribution of the file characterization. Next, the generator simulates several logins. Using a UNIX process, each login is simulated with specific file usage characteristics (i.e., average APB, average file size and number of files) that are taken from the user characterization. Actual read and write calls are issued to the simulated files, according to the distributions of the file characterization measures of the user type (heavy or light). Apart from recreating the measured file access characteristics, the generator can also produce a heavy or a light file access load by selecting a certain ratio of users from various categories (i.e. light, heavy, and so on). The information on sharing among users (via notes and user files) and file I/O rate is also useful in making the synthetic workload realistic. This synthetic workload is being used to evaluate file system performance and to evaluate some of the new policies discussed below.

B. Towards File System Design

Our study shows that the user-owned files are almost always completely referenced, but many notes files are rarely referenced in entirety, and they are quite large. These results suggest the use of different prefetch policies for different file categories. The fact that there is a large variability in file size may have some implications for networked systems also. These results suggest the use of file transfer protocols that can efficiently transfer small amounts (few tens of bytes) as well as large amounts (few ten thousands of bytes), which is unlike, for example, TCP/IP.

This study also shows that only user-owned read-write files and heavy users' files are also likely to be referenced heavily. The heavy referencing suggests a limited use of a delayed write-back policy for these classes of files. Since regular write-backs can be a source of response time degradation (particularly, during heavy usage periods), such a policy coupled with recent improvement in memory reliability can be considerably beneficial. Further, the results point towards a way to improve the file replacement policy by combining the LRU policy with a selection criterion based on the category of a buffered file and the current status of its user (heavy or light). Such a replacement policy may increase file buffer hit ratios, without significantly impairing the response to other files and users, since our results show that these other files are unlikely to be referenced more than once.

The results on file size show that 80% of files are 10K bytes or smaller, implying that the translation of a file name into an inode number can be an important performance issue (as it was also pointed out in [Floyd 86a]) for the measured system. It can be easily addressed with a small cache of name-to-inode mappings (as it is done in Version 4.3 of the Berkeley UNIX, and in [Floyd 86b]). Further, since an average user-owned directory is even smaller than 1024 bytes, a per-user directory cache of a few kilobytes might capture most references to directories.

The results on sharing may have some additional implications to how notes files are implemented in networked systems. It was observed that a typical user does not access notes files heavily (APB is about 0.54), but a few (about 20% of) notes files are extensively shared (simultaneous and otherwise).

These results suggest that the notes files, instead of being duplicated or buffered at each node, should probably be supported using centralized servers similar to what is done with the password files in SUN Microsystems UNIX.

It should be noted that the Berkeley UNIX [Quarterman 85] addresses some, but not all the issues raised here. For example, from Version 4.2 onwards, Berkeley UNIX uses a large disk block size to improve file reads from a disk, and a sophisticated scheme to avoid disk space fragmentation [Mckusick 85] that could result from a large disk block size. As a policy, UNIX uses only a single block read-ahead [Ritchie and Thompson 78] (4.2 and 4.3 BSD versions only make the implementation efficient), and in that way, UNIX deals somewhat with the uncertainity of whether a file will be referenced in entirety or not. It is worthwhile to examine how these schemes compare with what we suggest here in future networks that may consist of 100's or 1000's of workstations as well as many superminis and file servers ([Devarakonda 85] and [Satyanarayanan 85]).

3.9. Summary and Conclusions

Based on the short term file access data collected from a 4.2 BSD UNIX, this study quantified a typical user's file usage in a login session and the usage of a typical file in all login sessions. This approach is a departure from the traditional way of analyzing file references without actually characterizing either a user or a file. Two characterization measures were employed: accesses-per-byte (which combines fraction of a file referenced and number of references) and file size. It was shown that this new approach distinguishes differences in files as well as users. The multi-stage gamma were shown to model the file usage measures, which implies that the user demands cannot be assumed to be a single-stage exponential in performance evaluation.

Files and users belonging to various categories (based on ownership, type of use, UNIX file type, and file I/O) showed significant differences in their usage characteristics. More than 50% of users referenced files owned by other users, and over 8% of all files were involved in such references. Some group programming efforts and system utilities (such as *finger*) are the reasons for this result.

Significant simultaneous sharing occurred only to notes files, and that too involved only about 3% of all notes files.

Finally, the file and user characteristics measured here have been used to generate a synthetic file access workload to evaluate file system design. Based on the differences in files and users, suggestions to improve file system performance were also made.

CHAPTER 4

AN ANALYSIS OF REFERENCES

This chapter describes an analysis of *file references* using system call level file usage data collected from a VAX/780 running 4.3 BSD UNIX. The data was collected in the second measurement of the system as described in Chapter 2. The data includes system calls to all files (user-owned, notes files as well as system-owned files) that were referenced during the measured period. Here, a few simple measures based on the notion of file reference are used to develop system-level file usage patterns, as opposed to user and file characterizations of the previous chapter. It also provides time-based measures, such as inter-reference times, which were not included in the user-oriented analysis. Thus, the objective of this analysis is to obtain results that complement those obtained from the user-oriented analysis.

4.1. Overview

A file reference starts with an open or a creat system call to a file and encompasses any subsequent reads, writes, or lseeks, and concludes with an explicit close system call or termination of the process that started the reference. Based on this notion of file access, distributions of the following measures are analyzed:

Fraction Referenced:	The ratio of the sum of bytes read and written from a file to its maximum size during a reference.
File Size:	Maximum size of a file during a reference.
File Reference Time:	Length of time for which a file reference lasts.
Number of References per File:	Number of references to a file in a day or in its lifetime, whichever is shorter.
File Inter-Reference Time:	The time between the starting points (open or creat calls) of two successive references to the same file.

In this analysis, a file is identified in a way similar to that in the user-oriented analysis: Inode and device numbers are used in conjunction with a usage number. Recall that, as inodes are re-used in UNIX, the usage number distinguishes different uses of an inode.

Since how a file is used in a reference depends on who initiated the reference as well as who owns the file, file references are first categorized based on these attributes, and then separate distributions of the measures are obtained for each category. In UNIX, however, a file owner as well as a reference initiator is a user, even though some initiators or owners are pseudo-users such as notes. Hence, user classes categorize reference initiators as well as file owners, and consequently categorize file references. The following user classes are used in this study:

System:	(SYSTM) Pseudo-users such as *root*, *bin*, and *uucp* that own and maintain the operating system files.
Nonprivileged Users:	(NPUSR) Most Humans.
Notes:	(NOTES) Pseudo-user that owns and maintains the notes file system.

Using these user classes, a category of file references is identified by the pair: *initiator-owner*. For example, the category of file references made by nonprivileged users to system-owned files is designated as NPUSR-SYSTM, and references initiated by system daemons (e.g. *ruptime*) to operating system files is designated as SYSTM-SYSTM.

We also make a distinction between references to temporary and permanent files. Temporary (TEMP) files are those created in */tmp* or */usr/tmp* directories. The rest of the files are considered permanent (PERM) files.

In this analysis, only regular files are considered. The reason for excluding the references to directories is as follows. In an analysis of directory references, one must include direct references (i.e., read or write calls) as well as indirect references (accesses made to directories while translating a file name into an inode number). But, since the implementation of file name translation can change

substantially from one version of UNIX to another, the number of bytes accessed from a directory in such an indirect access can also change. This implies that when (direct as well as indirect) references to directories are analyzed, using a measure such as the fraction referenced, the results can heavily depend on how the file name translation is implemented. Since, in this analysis, we want to obtain results that are independent of system implementation, the references to directories are excluded.

A major results from this analysis is that most files are completely accessed, if referenced at all. This result provides further substantiation to our use of accesses-per-byte in user-oriented analysis. Most referenced files are small, but about 30% of references to system-owned files are to very large files. Most file references lasted only for a short duration of time (median: 80 milliseconds). Median inter-reference time is about 2 to 3 orders of magnitude larger than the file reference time.

The remainder of this chapter is organized as follows: Sections 4.2 through 4.6 discuss distributions of the measures defined above. Section 4.7 summarizes this chapter.

4.2. Fraction Referenced

Fraction referenced is computed from the measured values of bytes read or written to a file in a reference, and the maximum size of the file during the reference. The distribution is shown in figure 4.2.

As indicated by the solid line of the figure, in about 90% of references the fraction referenced is either 100% or nearly 0%. An examination of the data revealed that almost all the 0% fractions are owing to references made to zero sized files. It may be noted that, in UNIX, zero sized files are frequently used as lock files -- the notes file system makes extensive use of this programming practice.

The figure also shows that in over 95% of references made by nonprivileged users to nonprivileged user-owned (permanent) files, fraction referenced is either 100% or nearly 0%. But, in the case of (nonprivileged user initiated) references to notes or system-owned files, such an access pattern is evident in only 60% to 65% of references. The remaining references are incomplete accesses. In the case of system-owned files, most of such incomplete accesses are to log and database files such as

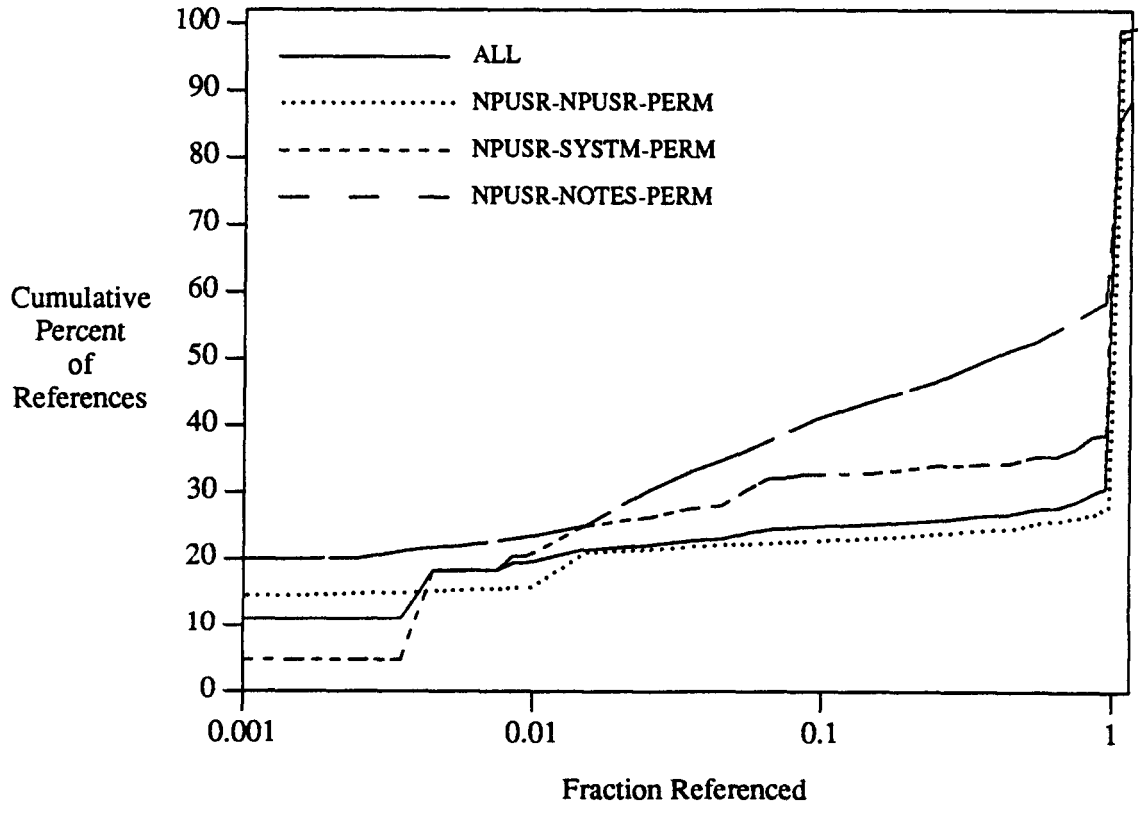


Figure 4.2: Distribution of Fraction Referenced

/etc/passwd.

In summary, fraction referenced depends on ownership of a file. In particular, 95% of references to user-owned files are complete accesses, whereas complete accesses occur in only about 60% of references to notes and system files.

4.3. File Size

File size, discussed in this subsection, is the maximum size of a file observed during a reference to the file. A distribution of such file sizes is often called dynamic file size distribution, because if a file is referenced several times during the measured period, then its size contributes as many times to the distribution. The distribution appears in Figure 4.3.

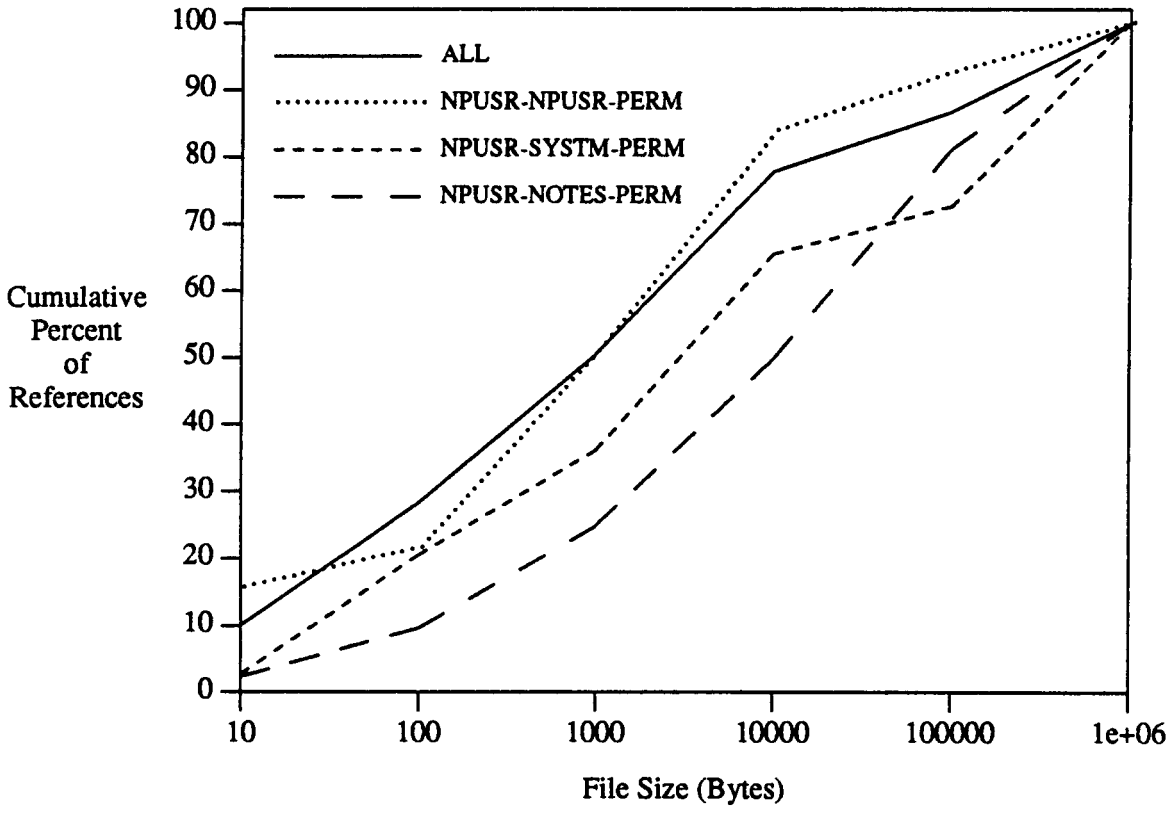


Figure 4.3: Dynamic File Size Distribution

Most references are to small files (median: about 1K bytes), as indicated by the solid line of the figure. The figure also shows the dependence of this measure on file ownership. For example, a median referenced file belonging to a nonprivileged user is about 1K bytes, but a median referenced file belonging to notes is a magnitude larger (10K bytes). A median referenced file belonging to the operating system falls between the two categories. However, about 30% of NPUSR-SYSTM references are to files that are larger than even 100K. An examination of the raw data revealed that these references are to system-related databases such */etc/termcap*.

In summary, a median referenced file is 1K bytes long. Dynamic file size is dependent on file ownership. About 30% of references to system-owned files are to files that are over 100K bytes long.

4.4. File Reference Time

File reference time is the time difference between an open or creat call to a file and a corresponding close system call or the terminating time of the process that started the reference, which ever is earlier. Thus, this measure shows how long a file reference lasted. File reference time distribution is shown in Figure 4.4.

Most file references are short. Median reference time for the general population as well as for references to system or user-owned files is only about 0.08 seconds. About 90% of file references are finished within 10 seconds of the starting time. As expected, however, references to notes files lasted relatively longer -- only 52% of these references are shorter than 10 seconds. Also, the distribution shows a distinct grouping of NPUSR-NOTES-PERM references based on how long a reference lasted: One group with less than 0.1 seconds reference time, and a second group of references (about 45%) that

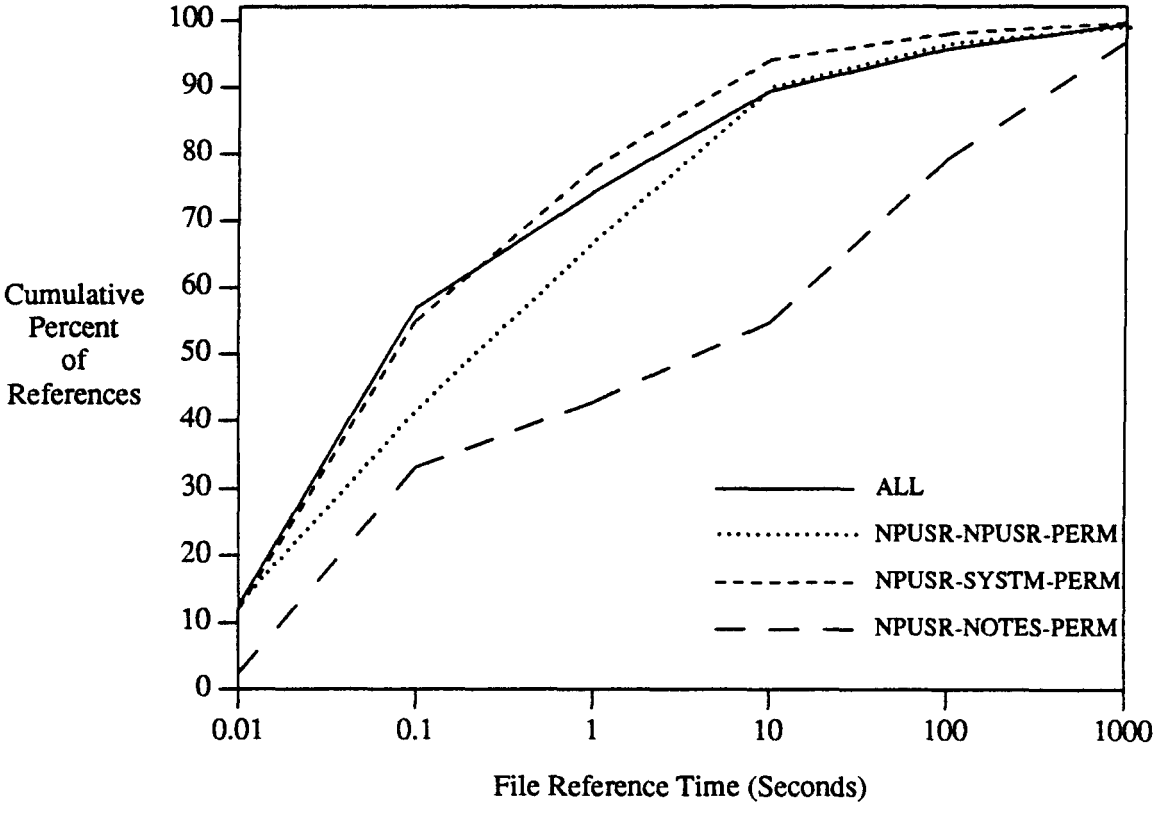


Figure 4.4: Distribution of File Reference Time

have long reference times lasting from 10 seconds to over 15 minutes. The first group corresponds to references made to lock files and inactive notes files, and the second group involves references to files containing actual bulletin-board messages.

In summary, most references lasted for a short time, with a median of about 0.08 seconds. But, about 45% of references to notes files ranged from 10 seconds to 15 minutes.

4.5. Number of References per File

This subsection discusses the distribution of the number of references per file, which is shown in Figure 4.5. This measure shows how often a file is referenced in a day or in its lifetime, which ever is shorter. The solid line of the figure shows that most files are referenced once or twice: 62% of files are referenced once, and another 25% are referenced twice. However, as the long tail of the distribution

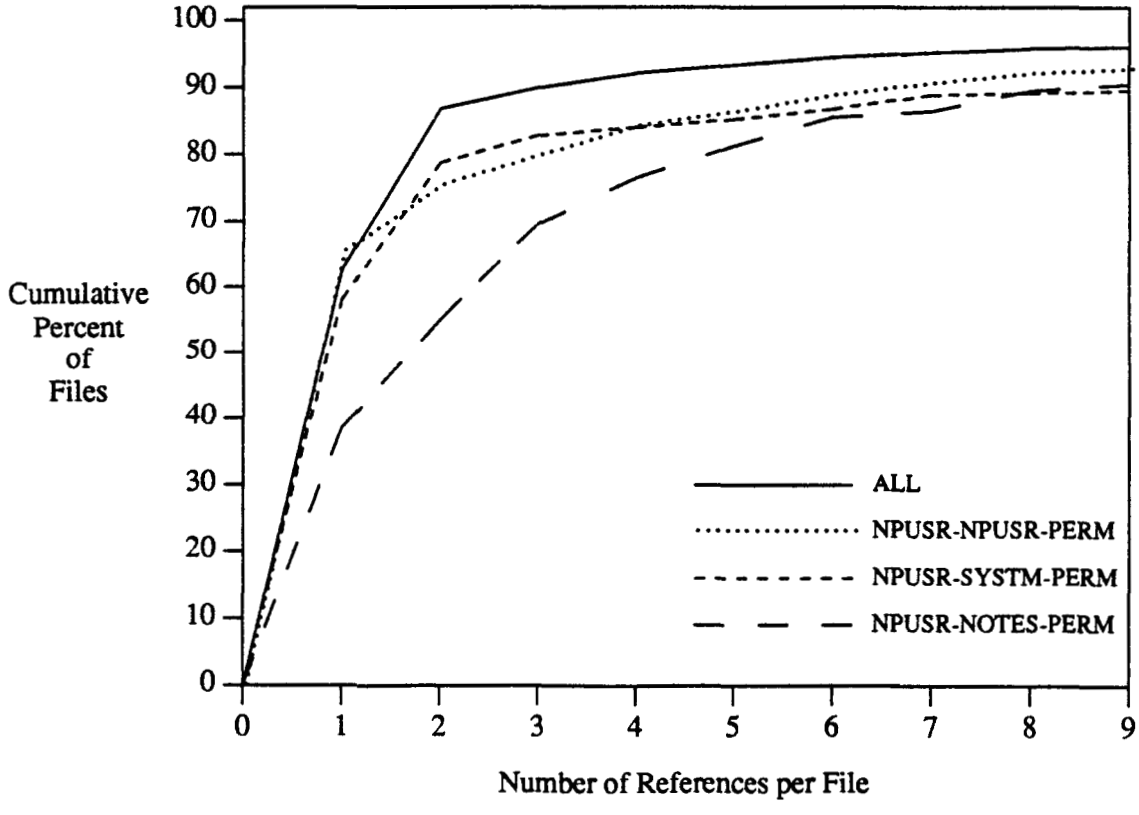


Figure 4.5: Distribution of Number of References

indicates, about 5% of files are referenced even more than 9 times. The figure also shows that files belonging to nonprivileged users, system, or notes are referenced more often than the general population. This is because the general population is influenced by short-lived temporary files, which are never referenced more than twice in their life-time.

On an average, as expected, notes files are referenced most often. But, as indicated by the tails of the distributions, files with 9 or more references are in equal percentage (about 15%) in NOTES and SYSTM categories of references.

In summary, most files are referenced infrequently. To a large extent, the number of references to a file depends on its ownership, but in almost every category a small percentage of files are referenced quite frequently (e.g., 15% of system-owned files are referenced 9 or more times).

4.6. Inter-Reference Time

In this subsection, we discuss distribution of time intervals between the starting points of two successive references to a file. Recall that the starting point of a reference is either an open or creat system call. The distribution is shown in Figure 4.6.

The distribution shows that median inter-reference time, which is about 45 seconds for the general population, is far larger than median reference time (0.08 seconds). Note, however, that inter-reference time of the general population is influenced by references made by *ruptime daemon* (to system files) to maintain load information about other systems on the local network. The daemon updates some of this information once in every 60 seconds and the rest once in every 3 minutes: Notice a 15% jump in the solid curve at 60 seconds, and another 10% jump at 180 seconds.

Among NPUSR initiated references to PERM files, inter-reference time is the smallest when a file belongs to SYSTM (median: 15 seconds), and the largest when it belongs to NOTES (median: 300 seconds). Inter-reference time for NPUSR owned files is between the two extremes, with a median of 60 seconds.

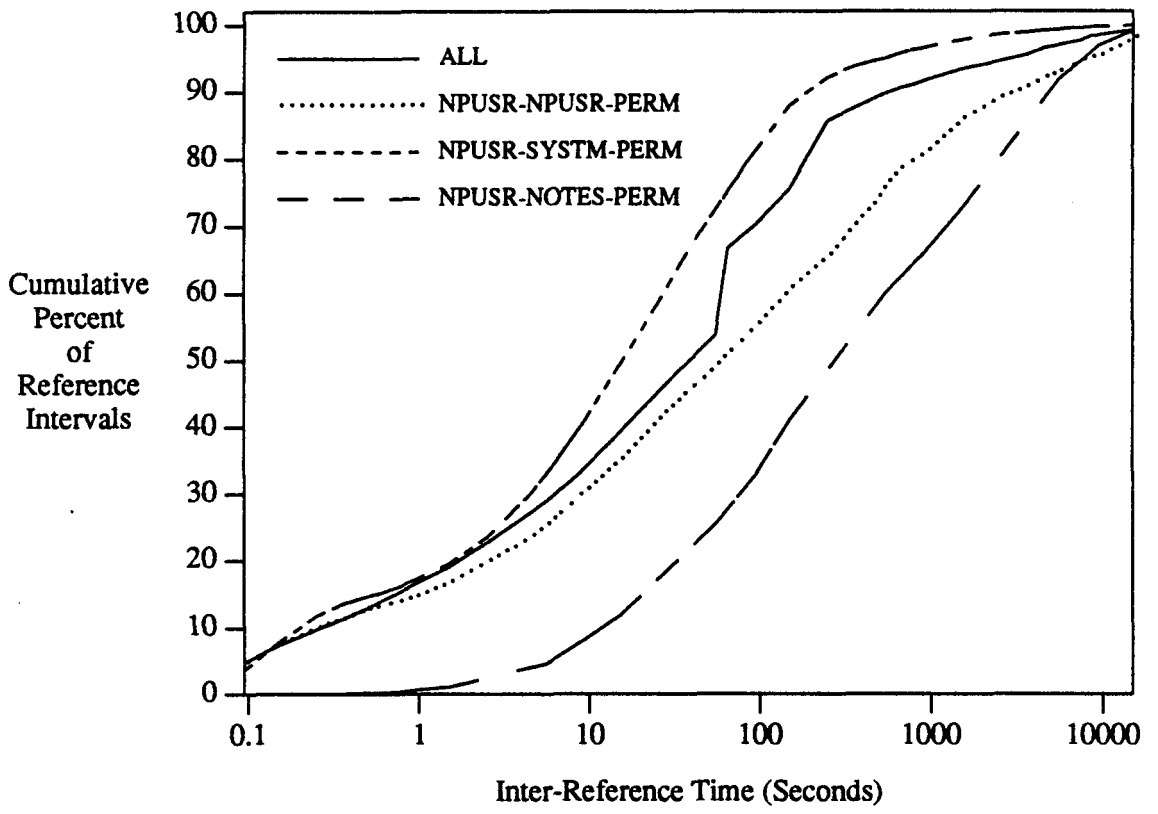


Figure 4.6: File Inter-Reference Time

In summary, median inter-reference time is 2 to 3 orders of magnitude larger than median reference time. If references made by nonprivileged users are only considered, system files have the shortest inter-reference time, followed by user-owned files and notes files (in that order).

4.7. Summary

In this chapter, file references were analyzed using access density measures (fraction referenced and number of references), a resource usage measure (file size), and time-based measures (reference-time and inter-reference time).

Some results of this analysis substantiated assumptions that were made in user-oriented analysis, and others strengthened results from it. It was shown that in most references, files were accessed completely, if accessed at all. For user-owned files, the probability of complete access is particularly high (about 0.95). This substantiates the argument for using access-per-byte measure in user-oriented

analysis.

In general, access patterns were shown to depend on file properties such as file ownership. For example, only about 60% of system-owned files are completely accessed in a reference, and inter-reference time for system-owned files was the smallest (median: 15 seconds).

Even though most files were referenced infrequently, a significant percentage of files in almost every category of files are quite frequently accessed. These results strengthen the results obtained in user-oriented analysis, and suggest user-based and file-based schemes to identify heavily used files (to optimize file buffer management).

Further, time-based results were added to the body of knowledge about short-term file usage. It was shown that most file references lasted for a short time (median: 0.08 seconds), and that inter-reference time was 2 to 3 orders of magnitude larger (median: 45 seconds) than reference time. Inter-reference time distribution showed a skew caused by system activity that updates load information of other systems on the local network. Also, note that the numerical results for time-based measures can be highly dependent on hardware (speed) characteristics of the measured system.

CHAPTER 5

PROCESS RESOURCE USAGE PREDICTION

The study reported in this chapter addresses two questions: Is it possible to predict resource requirements of a process? And if so, how well can we predict the requirements? Resource usage prediction can be a sound basis for load balancing in a distributed computer system, because costs associated with frequent load information exchange or process migration can be avoided. An additional motivation is in the area of reliable distributed computing-- knowledge of resource commitments can be valuable in reorganization of a system under failure.

To our knowledge, there are no empirical studies that predict process resource usage using statistical methods. One relevant study is [Zhou 86b], which concluded that system load cannot be predicted based on load indices. The study, however, does not address predictability of process resource requirements.

5.1. Overview

Here, we develop a probabilistic scheme for predicting CPU time, file I/O, and memory requirements of a process at the beginning of its life, given the identity of the program being run. The scheme consists of building a state-transition model for each program to represent resource usage of the program in its previous executions, and a procedure for computing resource requirements for the next execution of the program based on this state-transition model. An off-line statistical clustering procedure is used to identify the resource regions where processes are likely to occur. These resource regions are the states of the state-transition model. The prediction scheme is shown to work using process resource usage data that was collected from a VAX 11/780 running 4.3 BSD UNIX as described in chapter 2.

We quantified the quality of prediction in two ways: First, statistical correlation between the predicted and actual values are shown. Next, distributions of errors in prediction are plotted and characteristics of these distributions are discussed.

The results of our experiments show that the coefficient of correlation between predicted CPU time requirements and the actual values is 0.84. A perfect prediction would give a result of 1.0. The distributions of prediction errors are heavily skewed towards small values. That is, although there are a few large errors, most errors are small. For example, 82% of errors in CPU time prediction are less than 0.5 standard deviations. When contrasted with the large variability in process CPU times (the difference between 99 and 1 percentiles is about 18 standard deviations), the results are clearly good.

The organization of the remainder of the chapter is as follows: Section 5.2 discusses previous work related to this study. Section 5.3 describes basic statistics of process resource usage in the measured system. Section 5.4 describes resource usage modeling. Section 5.5 describes the prediction scheme in detail and provides error statistics. Section 5.6 examines issues such as the influence of varying the amount of past used in prediction on prediction error. Section 5.7 summarizes the chapter.

5.2. Background

In this section, we discuss desirability of resource usage prediction for load balancing purposes. We do that by comparing the resource usage prediction with other empirically observed, process or system, behavior as a basis for load balancing. Many load balancing algorithms have been proposed (for example, [Hwang 82; Bryant and Finkel 81]) and many more simulation studies have been made [Eager 86; Barak and Litman 85; Wang and Moris 85]. But, only two measurement-based load balancing schemes have appeared so far.

The first of such load balancing schemes [Leland and Ott 85] proposes a heuristic algorithm based on an empirically observed linear relationship between the residual CPU time of a process and its age. The heuristic approximates to a *spiral assignment* of processes. Assuming that the processes are ordered by the their age, the spiral assignment assigns process i to processor $i \ mod \ N$, where N is the

total number of processors. Although average residual CPU time requirements of processes can be predicted based on age (as the authors claim), such a prediction may not hold for a single process.

The second load balancing scheme [Zhou 86a] is actually a family of algorithms that gather or propagate (depending on whether the algorithm is centralized or decentralized) load information about a distributed system, and use that information to assign a new job to a processor in such a way that it reduces process response time. In a related study [Zhou 86b], Zhou also showed that process response time strongly depends on processor load, and that the CPU and I/O queue lengths are good indicators of the load.

Using trace-driven simulations, these load balancing schemes were shown to reduce process response times. But, the improvements are sub-optimal. Leland and Ott's load balancing algorithm performs poorly even without process migration. Zhou's algorithms rely on rapid and regular propagation of the global system status to all processors. Since costs associated with frequent exchanges of load information or process migration can be substantial, proper initial placement of processes based on predicted resource requirements of the processes is particularly attractive.

In [Zhou 86b], Zhou considered load indices as predictors of future system load, and he concluded that the future system load cannot be predicted based on the load indices. However, neither he nor any other measurement-based study ever addressed predictability of process resource requirements. This study proposes a probabilistic scheme to predict process resource requirements and shows that the scheme works on a trace collected from a production system.

5.3. Basic Statistics

In this section, we discuss distributions of process resource usage and inter-arrival times. These statistics characterize the measured system and bring out the variability in process resource usage; the latter shows the inherent difficulty in predicting the process resource usage.

Figures 5.3.1 through 5.3.4 show the cumulative distributions of process CPU time, file I/O, memory usage and inter-arrival times. Most processes used only a small amount of CPU time (median

0.24 seconds), but there are processes that used up to 33 minutes of CPU time. This large variability in process CPU times is also apparent from the fact that the standard deviation is over 13 times larger than the mean, and that the mean is larger than the median by a similar ratio.

File I/O distribution, Figure 5.3.2, shows that about 30% processes have accessed no file bytes at all, and that the distribution has several abrupt slope changes (for example, one such change can be seen just before the 10K bytes mark). As will be seen later, these characteristics make file I/O prediction harder than CPU time prediction.

Memory usage distribution, Figure 5.3.3, shows that most processes used only a small fraction of memory available on the system (median memory usage is 50K bytes). The distribution also shows the smallest amount of variability. Mean is less than twice as large as median, and the ratio of standard deviation and mean is about the same. These characteristics of the processes make memory usage prediction easier than CPU time prediction.

Even though the process inter-arrival time is of little consequence to the prediction scheme itself, we discuss its distribution to complete the understanding of the measured system. As can be seen from Figure 5.3.4, mean and median inter-arrival times are larger than the corresponding statistics of process CPU times. It implies that on an average the system utilization is not very high. However, since there are processes requiring large CPU times and small inter-arrival times, the system can be seen to have heavy as well as light usage periods.

In summary, resource usage distributions show that process CPU times have a large variability and that the system had a low as well as a high degree of utilization.

5.4. Resource Usage Modeling

In this section, we develop a state-transition model to describe dynamics of resource usage in a series of processes. Here, three resource usage parameters -- CPU time, file I/O, and memory used -- define a 3D resource space, and the processes that ran on the system (during an interval of time) are represented by points in the 3D space. A statistical clustering algorithm is employed to identify the

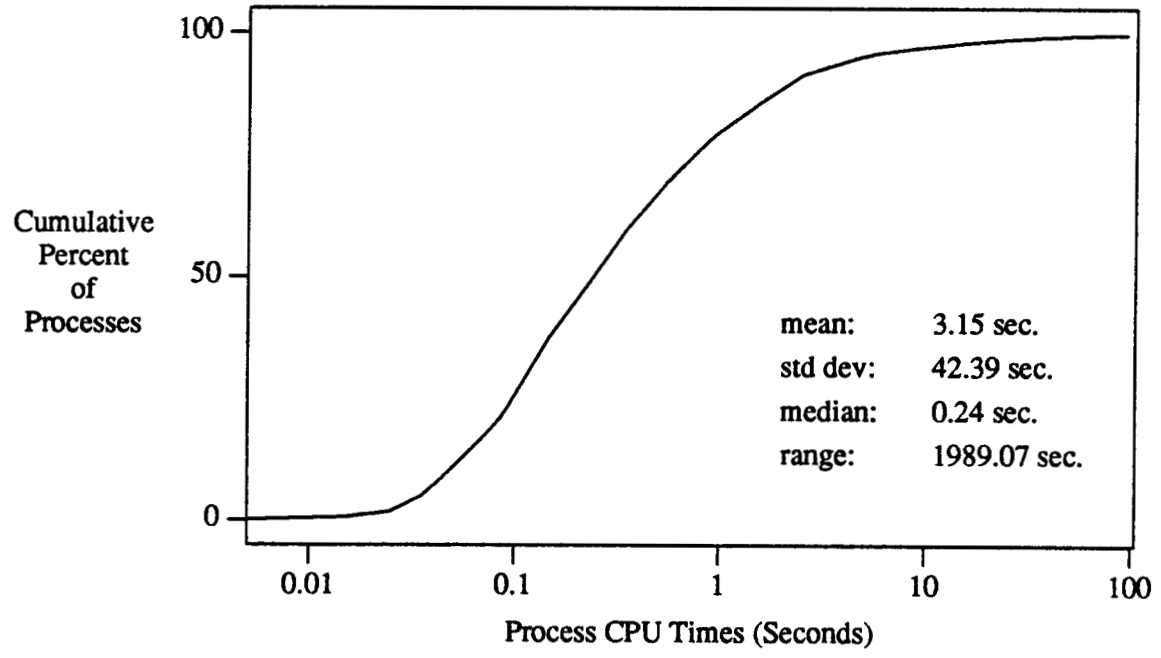


Figure 5.3.1: Distribution of Process CPU Times

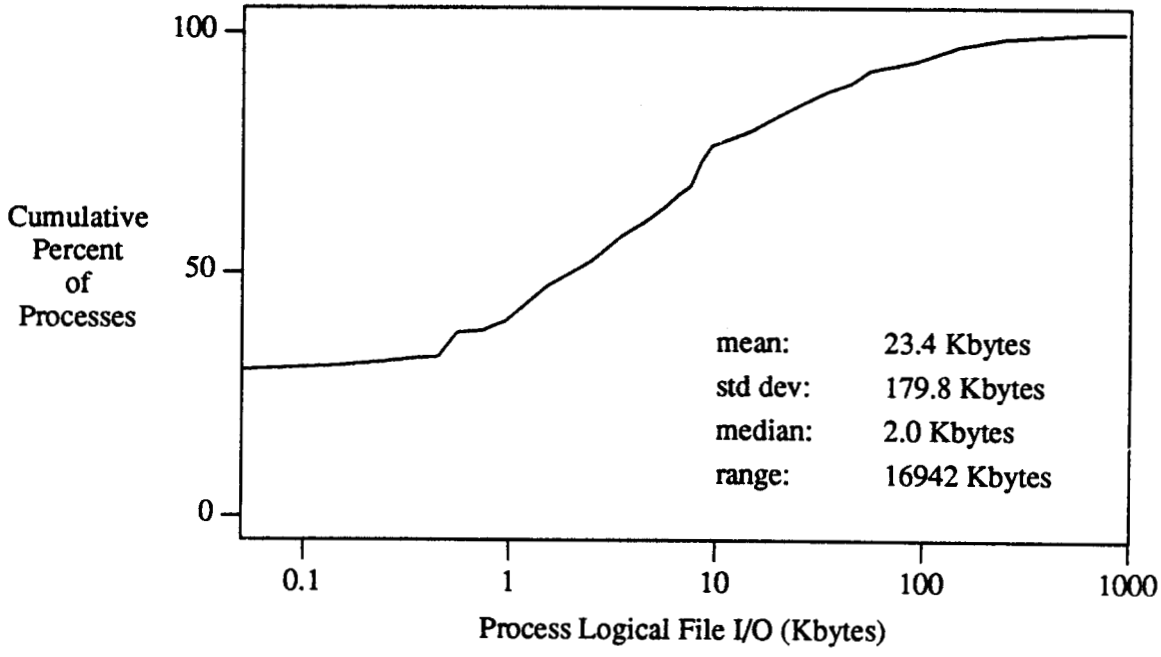


Figure 5.3.2: Distribution of Process File I/O

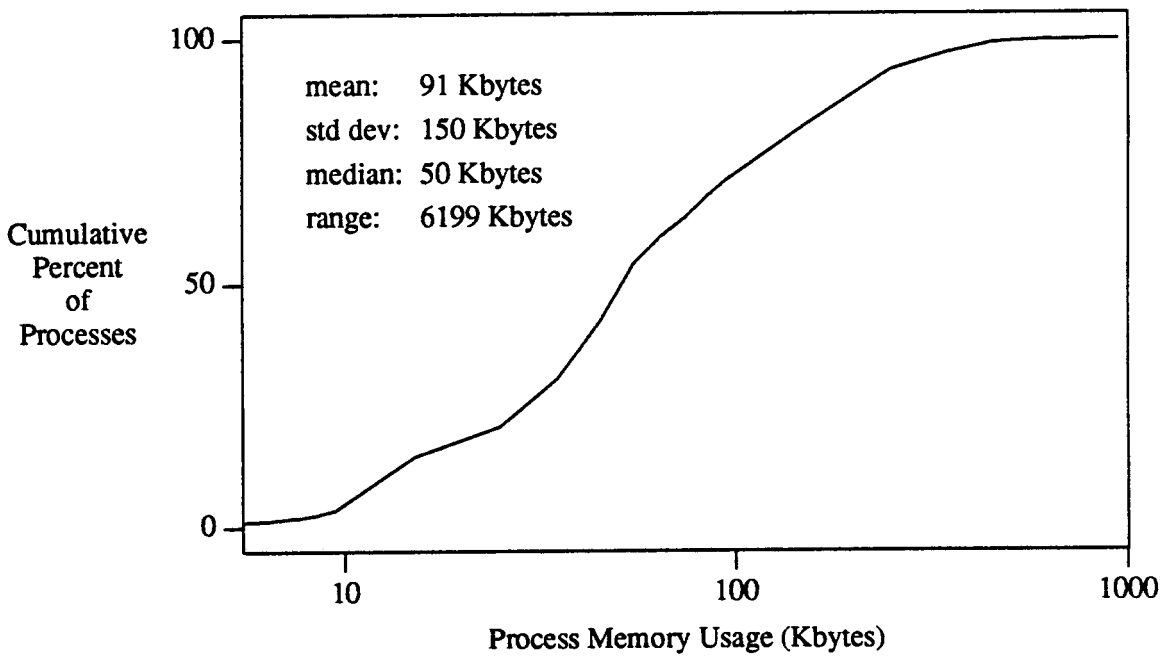


Figure 5.3.3: Distribution of Process Memory Usage

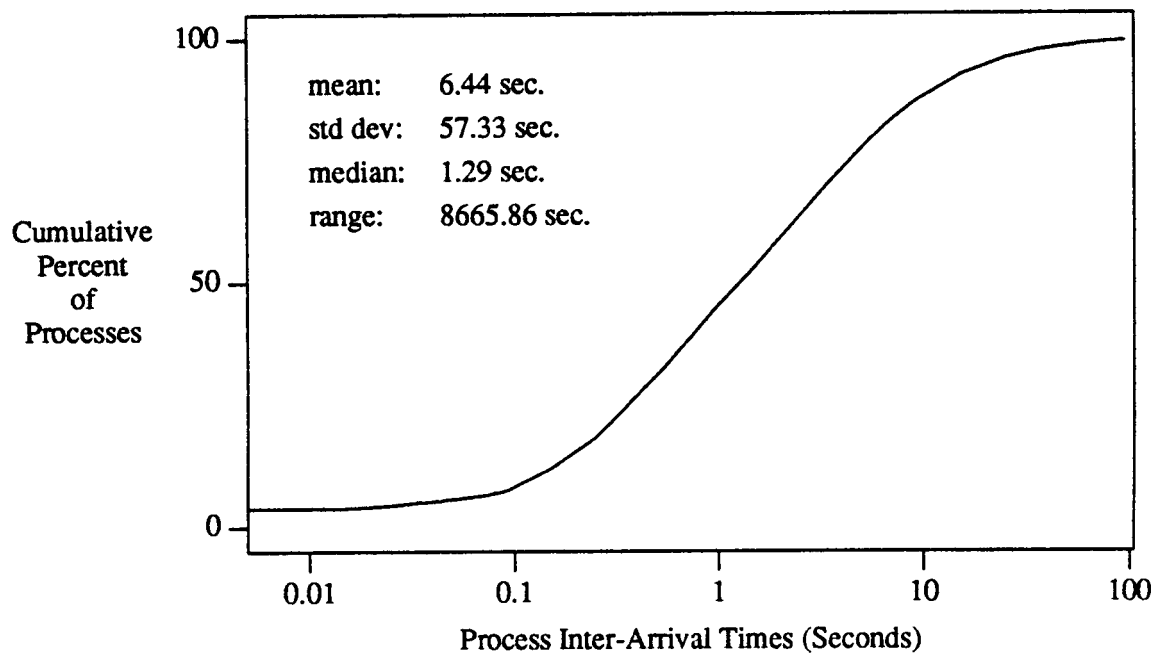


Figure 5.3.4: Distribution of Process Inter-Arrival Times

high density clusters in this space. These clusters, defined by their centroids, are taken to be the states for the processes, and appropriate transition probabilities are determined from one state to another. Later, this state-transition model will be used for representing the past resource usage, which in turn will be used to predict the future resource requirements.

5.4.1. Cluster Analysis

First, each of the three resource usage parameters are normalized so that the values are expressed in standard deviations rather than units specific to a resource. The normalization employed here is called z-transformation:

$$z_i = \frac{x_i}{\sigma_d} \qquad \text{(Eq.5.4.1)}$$

where z_i is the normalized value of x_i, and σ_d is standard deviation of the population with the largest d% of samples removed. We used d = 1.5 for CPU and file I/O and d = 0.5 for memory. The removal of the largest d% of samples eliminates the influence of the outliers on the normalization, and such a normalization can be helpful in obtaining well-defined clusters.

The cluster analysis used a k-means algorithm to partition an N-dimensional population into k clusters. Briefly, the algorithm starts with k clusters, each of which consists of a single random point. Each new point is added to the cluster with the closest centroid. After a point is added to a cluster, the mean of that cluster is recalculated to take the new point into account. The process is repeated several times, each time the initial means of k clusters are set to means from the end of the previous iteration, until the changes in the cluster means become negligibly small. Thus at any stage, the k means are in fact the means of the clusters they represent. Therefore, k non-empty clusters, $C_1, C_2, \ldots, C_k$, are sought such that the sum of squares of the Euclidean distances of the cluster members from their centroids is minimized, i.e.,

$$\textit{minimize} \quad \sum_{i=1}^{k} \sum_{J} |x_{ij} - \bar{x}_i|^2$$

where $x_{ij} \in C_i$ and $\bar{x}_i$ is the centroid of the cluster C_i.

Table 5.4.1: Cluster statistics.

Cluster Number	Cluster Frequency	Cluster Statistics (median values of the resources)		
		CPU (seconds)	File I/O (Kbytes)	Memory (Kbytes)
1	11.26%	4.62	13.870	194.726
2	2.64%	0.25	0.000	446.461
3	6.43%	0.80	8.486	192.444
4	9.42%	0.25	0.732	117.294
5	29.76%	0.07	0.000	16.000
6	29.69%	0.25	2.000	50.238
7	10.77%	1.54	103.804	134.386

Seven clusters of processes were formed. Table 5.4.1 shows the cluster statistics and percentage of processes in each cluster. We see from the table that clusters 1 and 7 represent heavy processes. Together they account for 22% of the population. Cluster 1 consists of CPU bound processes, and cluster 7 consists of balanced (CPU as well as I/O) processes. Another interesting class of processes belong to cluster 2: they are memory intensive.

5.4.2. State-Transition Model

Now that we have the clusters, we can calculate transition probabilities from one cluster to another to build a comprehensive state-transition model. A state-transition model built for a series of processes, taken from the measured data, is shown in Table 5.4.2 and in Figure 5.4.1. The processes are executions of a program. The transition probabilities from state i to state j, p_{ij}, were estimated using:

$$p_{ij} = \frac{observed\ number\ of\ transitions\ from\ state\ i\ to\ state\ j}{observed\ number\ of\ transitions\ from\ state\ i} \qquad \text{(Eq. 5.4.2)}$$

The state-transition model shows a distinct pattern. Transition probabilities from state 5 to itself (0.576) and from state 7 to itself (0.516), are the largest transition probabilities out of states 5 and 7 respectively. Note that the states 5 and 7 also have the highest *visit ratios* (see below). Therefore, from the model it can be concluded that an execution of the program is likely to be in state 5 or 7, and in addition, once an execution occurs in one of the states it tends to remain there. Patterns like these suggest predictability.

Table 5.4.2: A state-transition table for a program.

cluster#	1	2	3	4	5	6	7
1	-	-	-	-	-	-	-
2	-	0.250	-	-	0.250	-	0.500
3	-	-	-	-	-	-	-
4	-	-	-	0.410	0.205	0.154	0.231
5	-	0.003	-	0.038	0.576	0.050	0.333
6	-	0.018	-	0.036	0.382	0.109	0.455
7	-	0.003	-	0.031	0.357	0.093	0.516

Table 5.4.3: A visit ratio for a program.

cluster#	1	2	3	4	5	6	7
ratio	-	0.005	-	0.056	0.450	0.077	0.412

For some series of processes, however, transition probabilities out of a state are almost independent of current state. In such cases visit ratios are adequate. A visit ratio is the fraction of times a state occurred in a series of processes. For example, Table 5.4.3 shows visit ratios for the same series of processes that are used to build the state-transition model of Table 5.4.2. States 5 and 7 are visited 0.450 and 0.412 fractions of the time, making them the most frequently visited states. As will be seen in the next section, visit ratios, instead of transition probabilities, are used in prediction, when transitions to a state (and hence transitions out of that state) are too few to be statistically significant.

In summary, this section introduced a state-transition model for representing the dynamics of resource usage in a series of processes. The states of the model are the high density regions of a resource space, and they were obtained from a cluster analysis of the processes. We observed that the state-transition model can show interesting resource usage patterns.

5.5. A Program-Based Resource Prediction Scheme

Now that we have a state-transition model for representing the dynamics of resource usage in a series of processes, we describe how it is used for prediction. The particular scheme described here is a program-based prediction scheme. The scheme predicts resources required for a process at the start of

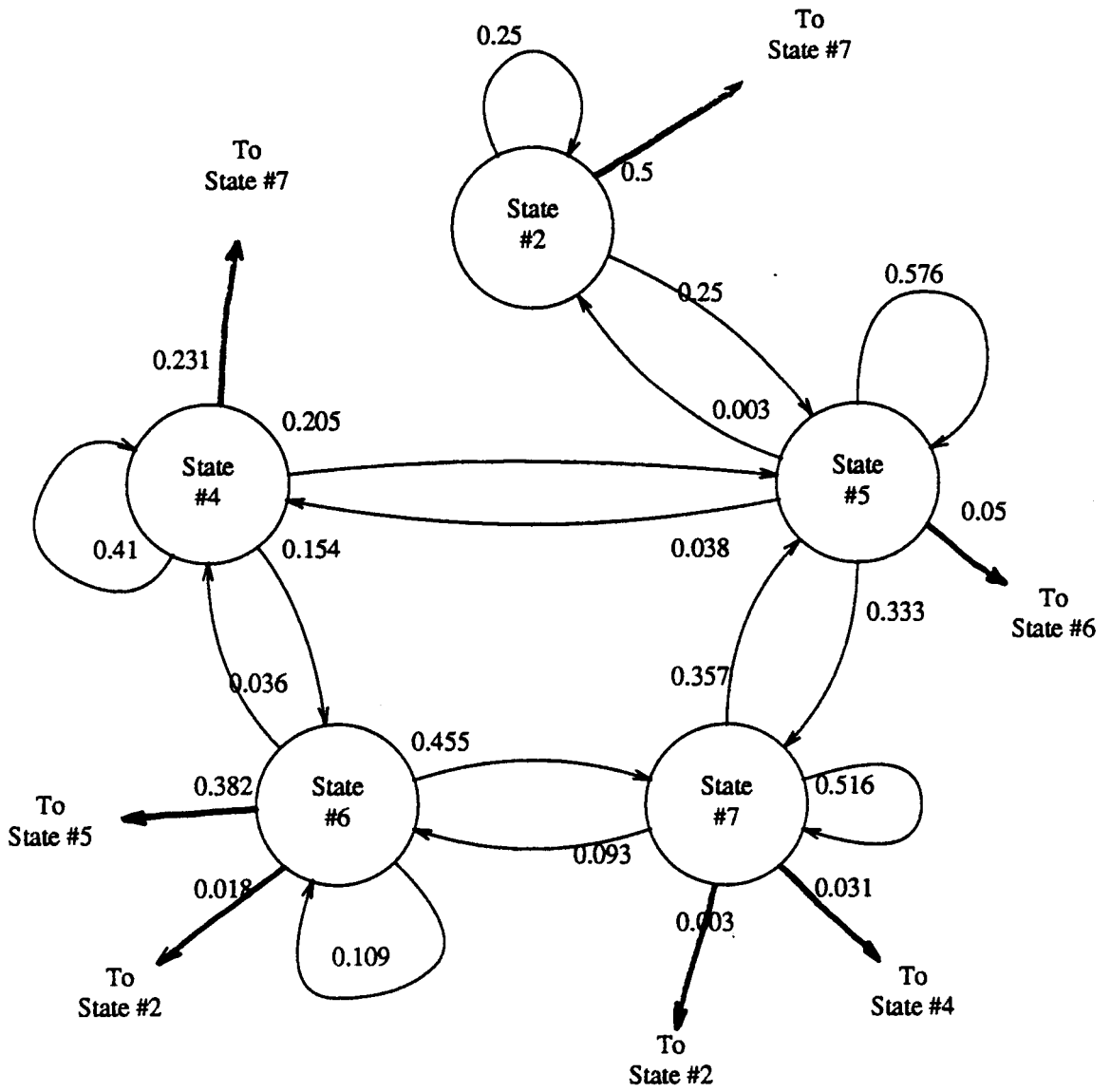


Figure 5.4.1: State-Transition Diagram for the Model in Table 5.4.2.

its life, given the identity of the program and resource usage of the program in its past executions. Hence, it is called program-based prediction.

The past executions of a program (for example, that of a LISP compiler) are ordered by the terminating times of processes, where the processes are the executions of the program. From this series a state-transition model, $[p_{ij}]$, $i=1,2,..,N$, $j=1,2,..,N$, is built using Eq. 5.4.2. Table 5.4.2 is an example of such a state-transition model.

There is an upper as well as a lower limit on the number of processes used in building the model. The upper limit, enforced via parameter T_1, restricts the amount of past used, and thus makes the model reflect a desired level of dynamic behavior. Of course, the exact number of past executions used is $min(m, T_1)$, where m is the number of past executions of a program that actually took place so far. In the implementation discussed here, we used all past executions of a program. The lower limit on the number of processes guarantees that the resource usage model is stable enough to make a prediction. Parameter T_2 of the prediction algorithm provides this lower limit.

Assuming that there are enough past executions, p_{lj}, $j = 1,2,...,N$, gives the probability that the next execution will be in cluster j, where l is the (resource usage) state of the program's previous execution. However, these transition probabilities are used in computing resource requirements only if the number of transitions out of the state l satisfy a minimum. Parameter T_3 represents this minimum, and it assures that the state has a statistically significant number of entries and exits. If this parameter is not satisfied, the prediction algorithm uses visit ratios (such as the ones in Table 5.5.3) for computing resource requirements.

The procedure for computing process resource requirements can be explained as follows. Since we have clustered the environment, each program execution must be in one of the clusters. Within each cluster, however, there is a subcluster that identifies the program. The midpoint of this subcluster is obtained by the most recent executions of the program that belong to the cluster. Then, the process resource requirements are obtained by multiplying the transition probabilities, p_{lj}, $j = 1,2,...,N$, with the

midpoints of these subclusters, d_{jk}, j=1,...,N, k=CPU ,I/O ,MEM:

$$r_k = \sum_{j=1}^{N} p_{1j} \times d_{jk}, \qquad k = CPU\ ,I/O\ ,orMEM$$

Note that d_{jk} are specific to a cluster as well as a program. A fourth parameter, T_4, determines the number of past executions used in computing d_{jk}. Also note that T_4 is considerably smaller than T_1. For example, in our implementation T_4=1, whereas T_1 is usually in the hundreds.

The prediction scheme is summarized in Figure 5.5.1. Parameter values used in our implementation of the scheme are shown in parenthesis. Now that we have described the prediction scheme, we will now proceed to discuss how well the prediction scheme worked on the data collected.

5.5.1. How Good is the Prediction?

In order to determine prediction quality, a trace-driven prediction experiment was conducted. The experiment consisted of predicting process resource requirements using the program-based method, just before the process started its life, and then observing the difference between the predicted and actual resource values after the process terminated. This section discusses results of this experiment.

For some processes prediction could not be made owing to the lack of enough past executions of the program. However, both the percentage of such processes and CPU time used by them are quite small. With T_1=3, less than 4% processes could not be predicted, and these processes used about 8% of CPU time.

We quantified prediction quality in two ways. First, product-moment (Pearson) and rank (Spearman) correlations [Mendenhall and Sincich 84] between the predicted and actual values are considered. The Pearson correlation coefficient measures the strength of the linear relationship between two quantities, and the Spearman's rank correlation measures correlation between ranks of the two quantities. Here, the Spearman's rank correlation is a better indicator than Pearson's because the former does not necessarily look for a linear relationship. Table 5.5.1 shows that the Pearson correlation coefficient is over 0.84 for CPU time and memory, but it is small (about 0.20) for file I/O. A correlation

Parameters:

T_1	Maximum number of past executions used in building the model (all).
T_2	Minimum number of past executions required to make a prediction (3).
T_3	Minimum number of visits to a state needed, to use the transition probabilities of the state ($max(T_2,\ 5\%\ of\ min(m, T_1))$).
T_4	Number of past executions used in computing subcluster centroids (1).

Constants:

N	Number of clusters (7).

Variables:

l	Cluster number to which the previous execution belonged.
m	Number of completed executions of the program so far.

Data structures:

$[p_{i,j}]$	State-transition matrix, $i = 1, ..., N$, and $j = 1, ..., N$.
$[v_i]$	Visit ratios, $i = 1, ..., N$.
$[s_{i,j,k}]$	Resources used in previous T_4 executions, $i = 1, ..., N$, $j = CPU, I/O, or\ MEM$, and $k = 1, ..., T_4$.
$[c_{i,j}]$	Cluster medians, $i = 1, ..., N$, $j = CPU, I/O, or\ MEM$.

Computations:

$$d_{i,j} = \begin{cases} \frac{1}{T_4}\sum_{k=1}^{T_4} s_{i,j,k} & \text{if } T_4 > 0 \\ c_{i,j} & \text{if } T_4 = 0 \end{cases} \qquad i = 1, ..., N, \text{ and } j = CPU, I/O, \text{ and } MEM$$

$$r_j = \begin{cases} \sum_{i=1}^{N} [\, p_{l,i} \times d_{i,j} \,] & \text{if } min(m, T_1) \geq T_3 \\ \sum_{i=1}^{N} [\, v_i \times d_{i,j} \,] & \text{if } min(m, T_1) < T_3 \end{cases} \qquad j = CPU, I/O, or\ MEM$$

Figure 5.5.1: Summary of the Program-Based Prediction Scheme.

Table 5.5.1: Correlations between Actual and Predicted Resource Values.

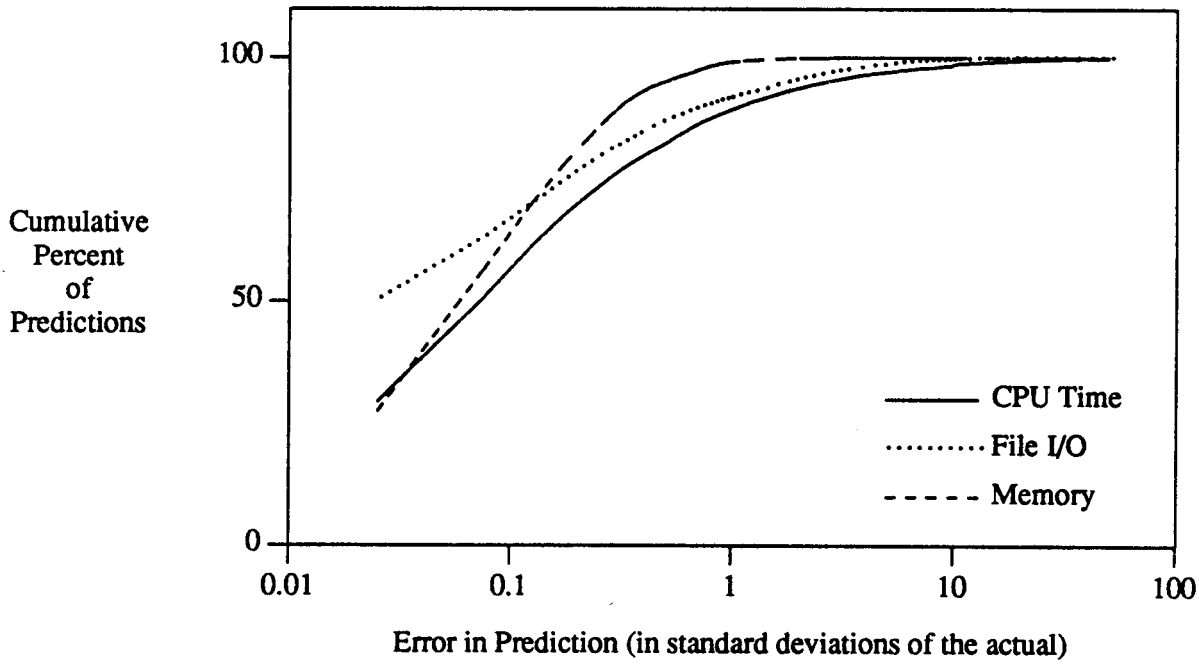
Resource	Correlation Coefficients	
	Rank (Spearman) Correlation	Product-Moment (Pearson) Correlation
CPU Time	0.8379	0.8406
File I/O	0.8105	0.1974
Memory	0.8925	0.8834

coefficient of 1.0 implies a perfect prediction. The Spearman correlation coefficient, however, ranges from 0.81 to 0.89 for all the resources. Clearly, quality of prediction is good.

Next, distributions of errors in prediction are considered. An error in prediction is the absolute difference between predicted and actual resource usage. Figure 5.5.2 shows distributions of prediction errors for CPU time, file I/O, and memory usage. It can be seen that error distributions are highly skewed towards small values. For example, 82% of errors in CPU time prediction are less than 0.5 standard deviations. Also, error in predicting memory usage is the smallest.

Mean and other statistics about prediction errors and actual resource usage values are shown in Table 5.5.2. The values are in normalized units (standard deviations of the actual) obtained through the application of z-transformation of Eq. 5.4.1. The table shows that for CPU time the median error is 0.073 standard deviations (about 43% of the actual), and the mean error is 1.224 standard deviations (about 53% of the actual). Since the variability in CPU times is large (about 18 standard deviations), as shown by the difference between 99 percentile and 1 percentile, we believe that these errors are acceptable.

Compared to errors in CPU time prediction, errors in file I/O prediction are larger, but errors in memory usage prediction are smaller. For example, median error in memory usage prediction is about 13% of actual, and mean error is about 19% of actual.



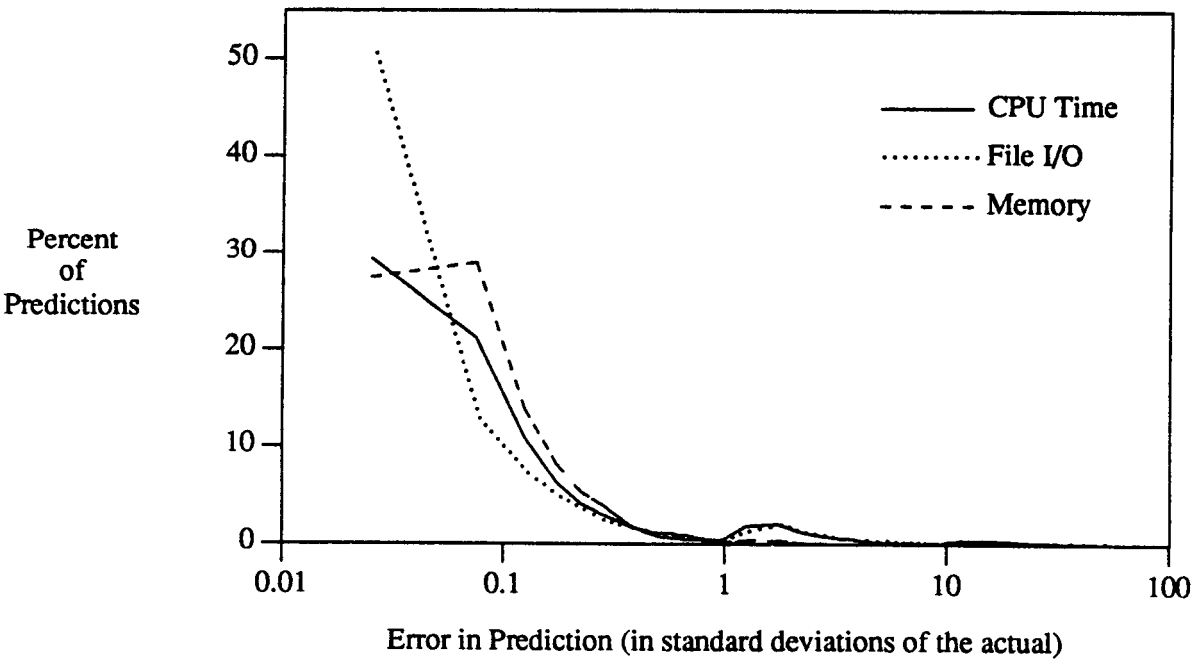


Figure 5.5.2: Distributions of Prediction Errors

Table 5.5.2: Statistics of the Prediction Errors and Actual Resource Values.

Resource		Statistic (in units normalized using Eq. 5.4.1)			
		Mean	Std dev	Median	99%-1%
CPU Time	Error	1.224	18.424	0.073	16.24
	Actual	2.230	32.780	0.168	18.23
File I/O	Error	0.485	4.909	0.024	6.13
	Actual	0.601	4.755	0.051	7.26
Memory	Error	0.140	0.560	0.059	0.97
	Actual	0.723	1.181	0.447	3.61

We considered other measures of prediction quality but rejected them on the grounds that they are not suited for the domain we are concerned with. For example, it might seem like a good idea to express the errors as percentages of the actual, and show a distribution of the percentages. However, (since the smallest amount of resource a process can use is 0) when a predicted value is smaller than actual, prediction error can be 0% through 100%, but when a predicted value is larger than actual prediction error is potentially unbounded. This distorted view of error can lead to a misleading perception that a scheme that makes a few large over-estimations is worse than a scheme that consistently underestimates.

We have also compared means and variances of predicted and actual values, and examined correlation between error and actual values. Means and variances of predicted and actual values match very closely. Errors correlate slightly positively (about 0.20) with actual values, implying that large prediction errors (if any) tend to occur only when outliers of process population occurs.

In conclusion, even though the program-based prediction scheme makes a few large errors, errors are mostly small.

5.6. Additional Implementation Issues

In the previous section, the program-based prediction was described in detail, and using a trace-driven experiment, it was shown that the error in prediction is small. Here, we discuss the following

three issues related to the implementation of the prediction scheme.

1. The influence of program execution frequency on prediction quality.

2. The influence of maximum and minimum past used in prediction on prediction quality.

3. The influence of system load on memory usage measurement.

5.6.1. The Influence of Program Execution Frequency

Each program is categorized as type 0, 1, 2, or 3 based on the total number of executions of the program during the measured period, and using a trace-driven experiment as described in the previous section, prediction quality is quantified for each program type. Results are shown in Table 5.6.1.

Type 0 consists of programs that are executed three or fewer times in the data, where 3 is the value used for the parameter T_2 (the minimum number of executions required to make a prediction). The

Table 5.6.1: Dependence of prediction quality on program execution frequency.

Item		Type #0 programs	Type #1 programs	Type #2 programs	Type #3 programs
Number of executions		1 thru 3	4 thru 8	9 thru 45	46 or more
Percent programs		36.4%	21.2%	21.0%	21.4%
Percent processes		2.7%	0.8%	4.4%	92.1%
Correlation of predicted and actual CPU times		-	0.803	0.794	0.879
CPU time statistics (in norm. units)	mean	-	19.971	13.629	1.531
	std dev	-	135.785	86.049	24.735
	median	-	0.488	0.595	0.160
Error in prediction (in norm. units)	mean	-	11.766	7.568	0.828
	std dev	-	90.537	54.498	11.935
	median	-	0.099	0.238	0.069
Error in prediction as pct of actual	mean	-	59%	56%	54%
	std dev	-	67%	63%	48%
	median	-	20%	40%	43%

remaining three types are defined such that the programs that are executed four (i.e., T_2+1) times or more are equally divided into the three types.

As can be seen from Table 5.6.1, about 36% of programs belong to type 0, and about 21% of programs belong to each of the remaining types. However, processes resulting from type 0 programs constitute only 2.7% of total processes. In comparison, processes resulting from type 3 programs are over 92% of the total. Programs of type 2 and 1 programs provide 4.4% and 0.8% processes each. Clearly, a small fraction of programs are executed frequently (e.g., 21% of programs are executed 92% of times).

For type 3 programs, the coefficient of correlation between predicted and actual CPU times is 0.879, and for types 1 and 2, the coefficient is about 0.8. A correlation coefficient of 1.0 implies a perfect prediction. Given that the observed correlations coefficients are above 0.8, prediction quality is quite good for processes produced by programs of any type. The prediction is particularly good for processes produced by type 3 programs, and these processes constitute a major fraction of processes that ran on the system.

Table 5.6.1 also shows statistics for process CPU times and prediction errors for each category of programs. The CPU times and errors are reported in normalized units obtained through the application of Eq. 5.4.1, so that these results can be easily compared with those reported in the previous section. The average CPU time used is the largest for processes resulting from type 1 programs, followed by processes resulting from type 2 programs. The average error in prediction follows the CPU time usage pattern. However, when expressed as a percentage of average CPU time used, the prediction error is comparable for all program types, with the error percentage being slightly higher for infrequently executed programs.

In summary, it is shown that the quality of prediction is essentially independent of program execution frequency, except for programs that are executed less than 4 times. These programs constitute about 36% of all executed programs, but produce only 2.7% of all processes. The next

section discusses how prediction quality varies when the maximum and minimum past used in prediction is varied.

5.6.2. The Influence of Maximum and Minimum Past Used

Here, we quantify the influence of maximum and minimum past used in the prediction scheme (parameters T_1 and T_2 of the prediction scheme) on quality of prediction.

A. Maximum Past Used

First, the trace-driven experiment described in the previous section is repeated several times, each time with a different value for the maximum past used in building the resource usage model, while keeping the minimum past fixed at 1. The mean error[1] in CPU time prediction, obtained from these experiments, is shown in Figure 5.6.1 for the maximum past ranging from 1 through 300.

The figure shows that the mean error decreases as the maximum past is increased. The rate of improvement saturates around a value approximately equal to 100. Note, however, that a change in the maximum past from 1 to 300 brings about a reduction of about 7% in mean error for CPU time prediction.

An examination of error distributions for different values of maximum past shows that when a small amount of maximum past is used (say $T_1 = 1$), the prediction is overly sensitive to local variations in the resource usage pattern of the predicted program. The error distribution for such a small maximum past (i.e., $T_1 = 1$) is more heavily skewed towards small values and has a longer tail than the error distribution for a large maximum past (say, $T_1 = 300$). Thus, when a large amount of maximum past is used, the prediction errors are evenly distributed while both large as well as small errors decrease. Consequently, using a large amount of maximum past (for example, 300) has a stabilizing effect on prediction, and results in a small average error.

[1]The error is shown in the same normalized units as the actual process CPU time, which is obtained using Eq. 5.4.1.

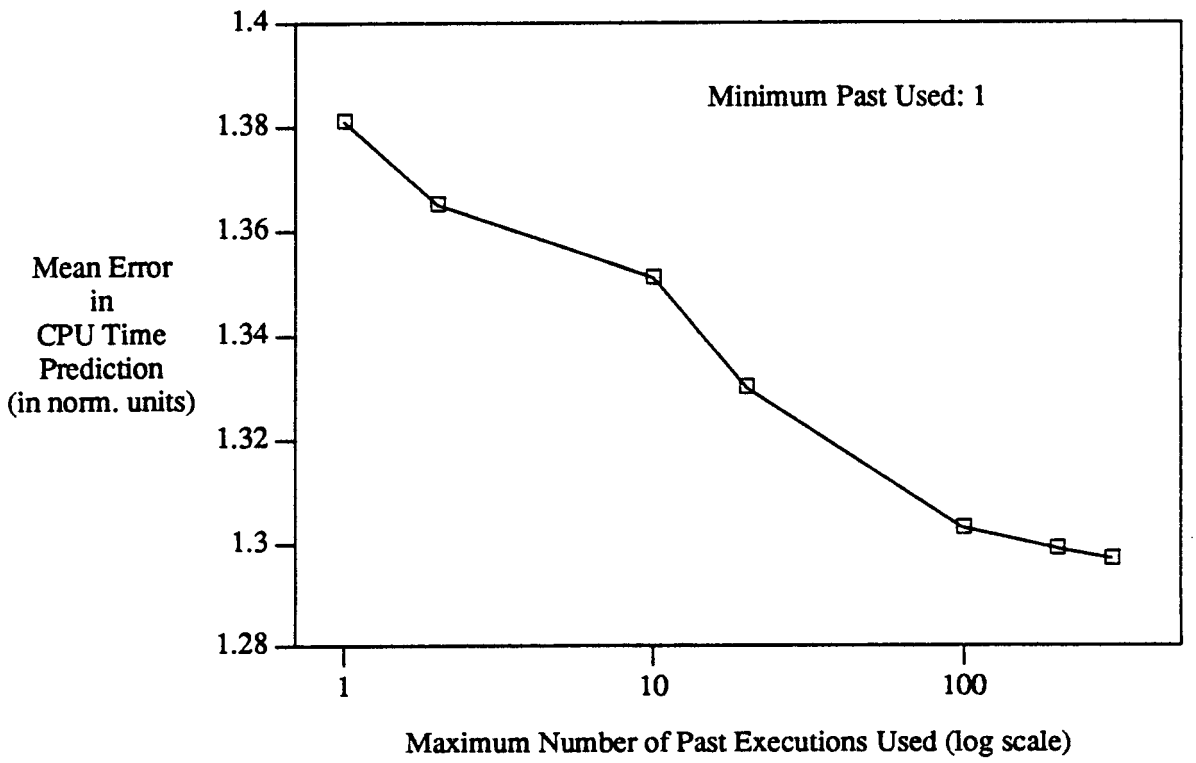


Figure 5.6.1: Effects of Changing Maximum Past Used in Prediction.

B. Minimum Past Used

Next, the effect of varying the minimum past used, parameter T_2, on prediction quality is examined. The trace-driven experiments are repeated once again with different values of minimum past, while keeping the maximum past fixed at 200. The results of these experiments are shown in Figure 5.6.2. The mean error[2] in CPU time prediction drops dramatically as the minimum past is increased -- the prediction error reduces by about 38% as the minimum past is changed from 1 to 20 executions.

However, unlike the changes in maximum past, increasing the minimum past has a side-effect of decreasing the percentage of predictable processes. More importantly, an increase in the minimum past decreases the percentage of predicted CPU usage by a considerable amount. For example, as the minimum past is raised from 1 to 20, the percentage of predicted processes drops by only 9%, but the

[2]The error is shown in the same normalized units as the actual process CPU times, which is obtained using Eq. 5.4.1.

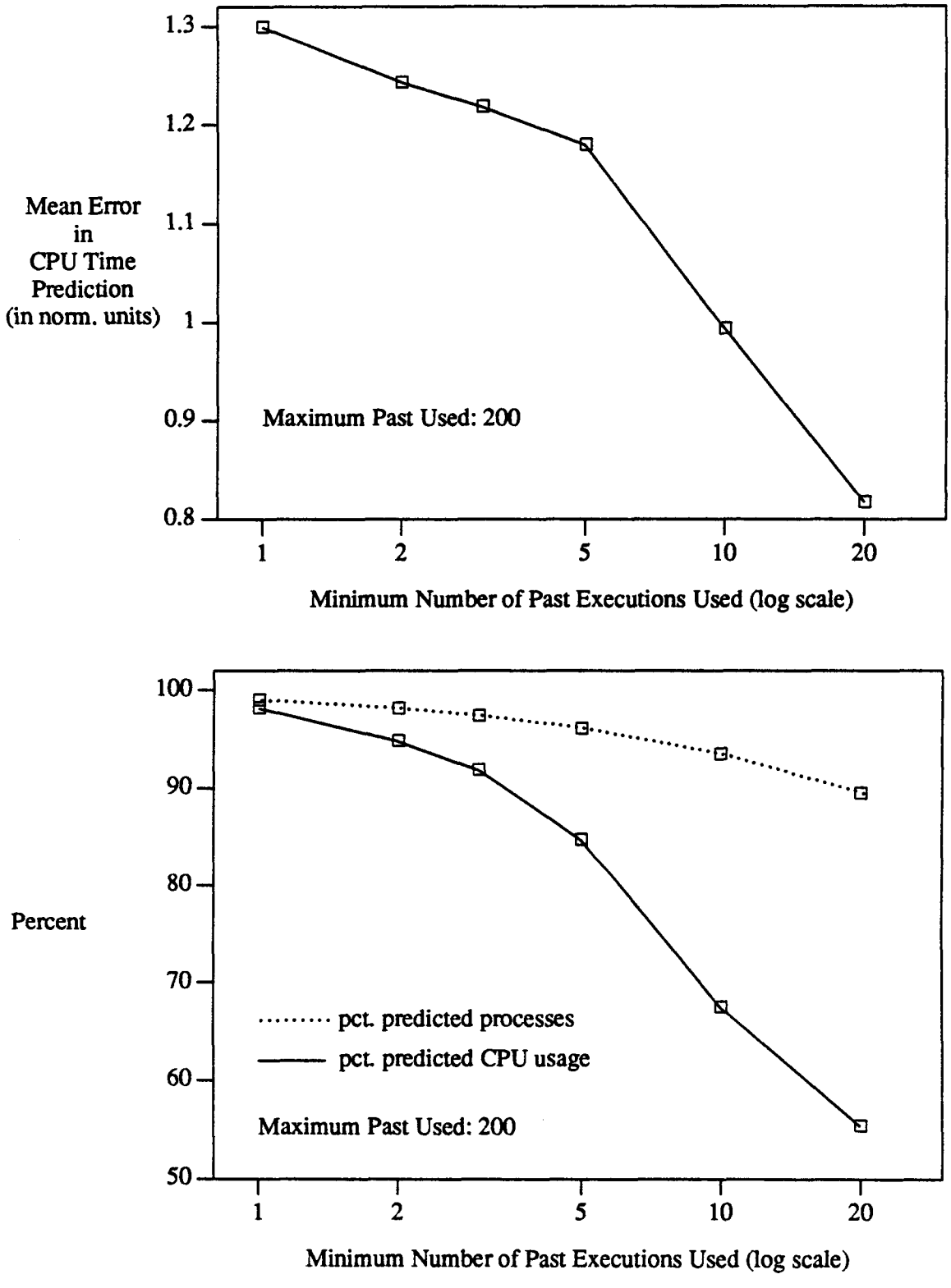


Figure 5.6.2: Effects of Changing Minimum Past Used in Prediction.

percentage of predicted CPU usage drops by 43%. So, a small minimum past, such as 3, is recommended.

5.6.3. System Load Influence on Memory Usage Measurement

The measured per process memory usage is the average amount of memory allocated to the process by the system. Since this allocation can depend on system *load*, we study the extent of such a dependency in this section. (The system load referred to here is the average number of ready-to-run processes on the system in the last one minute.) In order to do so, four programs, each with a different running time and memory usage pattern, were run on the measured system at regular (about 12 to 15 minutes) intervals for about two days, while the system was in normal use. For each execution of these programs, the system load and resource usages were recorded.

Based on these experimental measurements, we calculate the coefficient of correlation between the system load and memory usage, for each of the four programs. The results are shown in Table 5.5.2. As the table shows, for a long running program (e.g. 30 secs) having a small working set compared to its address space, the system load has the most prominent effect on the measured memory usage. The correlation coefficient for this type of program is -0.7824, indicating a negative correlation. However, for a program with a similar memory referencing pattern, but a shorter running time, the effect is not as

Table 5.6.1: Correlation between system load and process memory usage.

program characteristics		correlation coefficient	Is correlation statistically significant?
running time	memory usage pattern		
large (30 secs)	ws $\ll$ address space	-0.7824	Yes
small (3 secs)	ws $\ll$ address space	-0.4809	Yes
large (30 secs)	ws = address space	0.0435	No
small (3 secs)	ws = address space	0.2134	No

strong. For this type of program, the coefficient of correlation is only -0.48. Finally, for a program having the working set that is almost equal to its address space, independent of its running time, the system load influence on memory usage measurement is statistically insignificant.

The following, however, should be noted in this regard. Even when measurements are sensitive to system load, the resource usage model can incorporate these influences, and the prediction made using the model is valid if the target processor has a load similar to that of the measured processor. Since, the latter condition is likely to be true in a load balanced system, the influence of system load on memory usage measurement is not a serious problem.

5.7. Summary

In this chapter, we described a probabilistic scheme for predicting CPU time, file I/O, and memory requirements of a process at the beginning of its life. Given the identity of the program being run, this prediction scheme uses a state-transition model of the resource usage in the previous executions of the program. The states of the model are obtained from a statistical cluster analysis of the processes run on the system (in a day). The prediction scheme was shown to work on the measured data using a trace-driven prediction experiment.

The results of the trace driven experiment show that the predicted values correlate well with the actual. The coefficient of correlation between the predicted and actual CPU time is 0.84. Further, the error distributions show that the errors in prediction are mostly small. For example, 82% of errors in CPU time prediction are less than 0.5 standard deviations of process CPU time. These results are particularly interesting since Zhou's study [Zhou 86b] of system load indices as predictors of future load correlated poorly with the actual (correlation coefficients are always less than 0.45). Applications of resource usage prediction in load balancing and in system reorganization under failure are suggested as future work.

CHAPTER 6

SUMMARY AND FUTURE RESEARCH

This thesis demonstrated a practical methodology for file usage analysis and resource usage prediction using data collected from a production system.

6.1. Summary of the User-Oriented Analysis of File Usage

This analysis quantified a typical user's file usage in a login session and the usage of a typical file in all login sessions. This approach is a departure from the traditional way of analyzing file references without actually characterizing either a user or a file. Two characterization measures were employed: accesses-per-byte (which combines fraction of a file referenced and number of references) and file size. It was shown that this new approach distinguishes differences in files as well as users. The multi-stage gamma distribution was shown to model the file usage measures, which implies that the user demands cannot be assumed to be a single-stage exponential in performance evaluation.

Files and users belonging to various categories (based on ownership, type of use, UNIX file type, and file I/O) showed significant differences in their usage characteristics. More than 50% of users referenced files owned by other users, and over 8% of all files were involved in such references. Some group programming efforts and system utilities (such as *finger*) are reasons for this result. Significant simultaneous sharing occurred only to notes files, and that too involved only about 3% of all notes files. Based on the differences in files and users, suggestions to improve file system performance were also made.

6.2. Summary of the Analysis of File References

File references were analyzed using access density measures (fraction referenced and number of references), a resource usage measure (file size), and time-based measures (reference-time and inter-

reference time).

Results of this analysis substantiated assumptions that were made in user-oriented analysis. It was shown that in most references, files were accessed completely, if accessed at all. This substantiates the argument for using access-per-byte measure in user-oriented analysis. In general, access patterns were shown to depend on file properties such as file ownership. For example, only about 60% of system-owned files are completely accessed in a reference, and inter-reference time for system-owned files was the smallest (median: 15 seconds). It was shown that most file references lasted for a short time (median: 0.08 seconds), and that inter-reference time was 2 to 3 orders of magnitude larger (median: 45 seconds) than reference time.

6.3. Summary of the Resource Usage Prediction

In this part of the work, a probabilistic scheme for predicting CPU time, file I/O, and memory requirements of a process (at the beginning of its life) was described. Given the identity of the program being run, this prediction scheme uses a state-transition model of the program's resource usage in its previous executions. The states of the model are obtained from a statistical cluster analysis of the processes ran on the system (in a day). The prediction scheme was shown to work on the measured data using a trace-driven prediction experiment.

The results of the trace driven experiment show that the predicted values correlate well with the actual. The coefficient of correlation between the predicted and actual CPU time is 0.84. Error distributions show that the errors in prediction are mostly small. For example, 82% of errors in CPU time prediction are less than 0.5 standard deviations of process CPU time.

6.4. Future Research

Predictive Models for File Access: Results of the file usage analysis show that there are significant differences among files and users: some files are more heavily used than others, and some users access their files more heavily than the others. An interesting extension of this research is to develop statistical

models that can comprehensively represent the file usage patterns and predict future usage. These models can be based on users, on files, or on both.

Applications of the File Usage Analysis: Based on the differences in files and users, suggestions to improve file system performance were made in the user-oriented analysis. It would be interesting to implement these suggestions, and quantify the effectiveness of these new schemes over currently used methods. The file usage analysis also provides measures and distributions that can be used to evaluate the performance of a new file system.

Prediction-Based Load Balancing Algorithms: Predictability of process resource usage allows us to develop a new family of load sharing algorithms for multiprocessor systems. Compared to earlier schemes, these prediction-based algorithms will have less communication overhead (as system status need not be collected or propagated on a regular basis).

System Reorganization Under Failure: When a component of a multiprocessor system is failing, it is necessary to redistribute its load to other components, so that performability requirements are satisfied in the best possible way. This reorganization can be done easily, if resource commitments for each job are known. Resource usage prediction is valuable in such situations.

REFERENCES

[Barak and Litman 85] A. Barak, and A. Litman, "A Distributed Load Balancing Policy for a Multicomputer," *Software - Practice and Experience*, 15, 8, Aug. 1985.

[Barrington 86] T. Barrington, "A Synthetic Workload Generator Based on the User-Oriented Analysis of File Usage," *EE 441 Project Report*, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, 1986.

[Berkeley UNIX 84] *UNIX Programmers Manual: Reference Guide*, 4.2 Berkeley Software Distribution, Virtual Vax-11 Version, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720, March, 1984.

[Berkeley UNIX 86] *UNIX Programmer's Manual: Reference Guide*, 4.3 Berkeley Software Distribution, Virtual VAX-11 Version, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720, 1986.

[Box 78] G. E. P. Box, W. G. Hunter, and J. S. Hunter, *Statistics for Experimenters*, John Wiley & Sons, 1978.

[Bryant and Finkel 81] R. Bryant, and R. Finkel, "A Stable Distributed Scheduling Algorithm," *Second International Conference in Distributed Computing Systems*, IEEE Computer Society, Los Alamitos, California, April 1981.

[Daniel 78] W. W. Daniel, *Applied Nonparametric Statistics*, Houghton Mifflin Co., 1978.

[Devarakonda 85] M. Devarakonda, R. Mcgrath, R. Campbell, and W. Kubitz, "Networking a Large Number of Workstations Using UNIX United," *Proc. of 1st Intl. Conf. on Computer Workstations (IEEE)*, Nov. 1985, pp. 231-239.

[Eager 86] D. Eager, E. Lazowska, and J. Zahorjan, "Dynamic Load Sharing in Homogeneous Distributed Systems," *IEEE Transactions on Software Engineering*, SE-12, 5, May 1986.

[Essick 84] R. B. Essick IV, "Notesfiles: A UNIX Communications Tool," *Technical Report UIUCDCS-R-84-1165*, University of Illinois at Urbana-Champaign, Urbana, 1984.

[Floyd 86a] R. A. Floyd, "Short Term File Reference Patterns in a UNIX Environment," *Technical Report 177*, University of Rochester, March 1986.

[Floyd 86b] R. A. Floyd, "Directory Reference Patterns in a UNIX Environment," *Technical Report 178*, University of Rochester, March 1986.

[Georgiou 87] C. J. Georgiou, S. L. Palmer, and P. L. Rosenfeld, "An Experimental Coprocessor for Implementing Persistent Objects on an IBM 4381," *Second International Conference on Architectural Support for Programming Languages and Operating Systems*, Palo Alto, California, October 5-8, 1987.

[Hogg and Tanis 83] R. V. Hogg, and E. A. Tanis, *Probability and Statistical Inference*, Macmillan Publishing Co., 1983.

[Hwang 82] K. Hwang, W. Croft, G. Goble, B. Wah, F. Briggs, W. Simmons, and C. Coates, "A UNIX-based Local Computer Network with Load Balancing," *IEEE Computer*, 15, 4, April 1982.

[Johnson 87] T. D. Johnson, J. M. Smith, E. S. Wilson, "Disk Response Time Measurements," *Winter 1987 USENIX Technical Conference*, Washington, D. C., January 1987.

[Leland and Ott 85] W. Leland, and T. Ott, "Load-balancing Heuristics and Process Behavior," *Performance '86 and ACM SIGMETRICS Conference*, Raleigh, North Carolina, May 1986.

[Mckusick 84] M. K. Mckusick, W. J. Joy, S. J. Leffler, and R. S. Fabry, "A Fast File System for UNIX," *ACM Transactions on Computer Systems*, Vol. 2, No. 3, August 1984, pp. 181-197.

[Mendenhall and Sincich 84] W. Mendenhall, and T. Sincich, *Statistics for the Engineering and Computer Sciences*, Dellen Publishing Company, San Francisco, California, 1984.

[Ousterhout 85] J. Ousterhout, H. Da Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson "A Trace-Driven Analysis of the UNIX 4.2 BSD File System," *Proc. of Tenth Symp. on Operating Systems Principles*, Dec. 1985, pp. 35-50.

[Porcar 82] J. M. Porcar, "File Migration in Distributed Computer Systems," *Ph. D. Thesis*, University of California, Berkeley, CA, July 1982.

[Quarterman 85] J. S. Quarterman, A. Silberschatz, and J. L. Peterson, "4.2BSD and 4.3BSD as Examples of the UNIX System," *ACM Computing Surveys*, Vol. 17, No. 4 (Dec. 1985).

[Ritchie and Thompson 78] D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," *Bell System Tech. J.*, Vol. 57, No. 6, Part 2, July-August 1978.

[SAS 85a] *SAS User's Guide: Basics*, Version 5, SAS Institute Inc., Cary, NC 27511, 1985.

[SAS 85b] *SAS User's Guide: Statistics*, Version 5, SAS Institute Inc., Cary, NC 27511, 1985.

[Satyanarayanan 81] M. Satyanarayanan, "A Study of File Sizes and Functional Lifetimes," *Proc. of Eight Symp. on Operating Systems Principles*, Dec. 1981, pp. 96-108

[Satyanarayanan 85] M. Satyanarayanan, J. Howard, D. Nichols, R. Sidebotham, A. Spector, and M. West, "The ITC Distributed File System: Principles and Design," *Proc. of Tenth Symp. on Operating Systems Principles*, Dec. 1985, pp. 35-50

[Smith 81] A. J. Smith, "Analysis of Long Term File Reference Patterns for Application to File Migration Algorithms," *IEEE Trans. on Software Engineering*, Vol. SE-7, No. 4 (July 1981).

[Wang and Moris 85] Y.-T. Wang, and R. Morris, "Load Sharing in Distributed Systems," *IEEE Transactions on Computers*, C-34, 3, March 1985.

[Zhou 86a] S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing," *Tech. Report No. UCB/CSD 87/305*, University of California, Berkeley, California, Sept. 1986.

[Zhou 86b] S. Zhou, "An Experimental Assessment of Resource Queue Length as Load Indices," *Tech. Report No. UCB/CSD 86/298*, University of California, Berkeley, California, Sept. 1986.

APPENDIX A

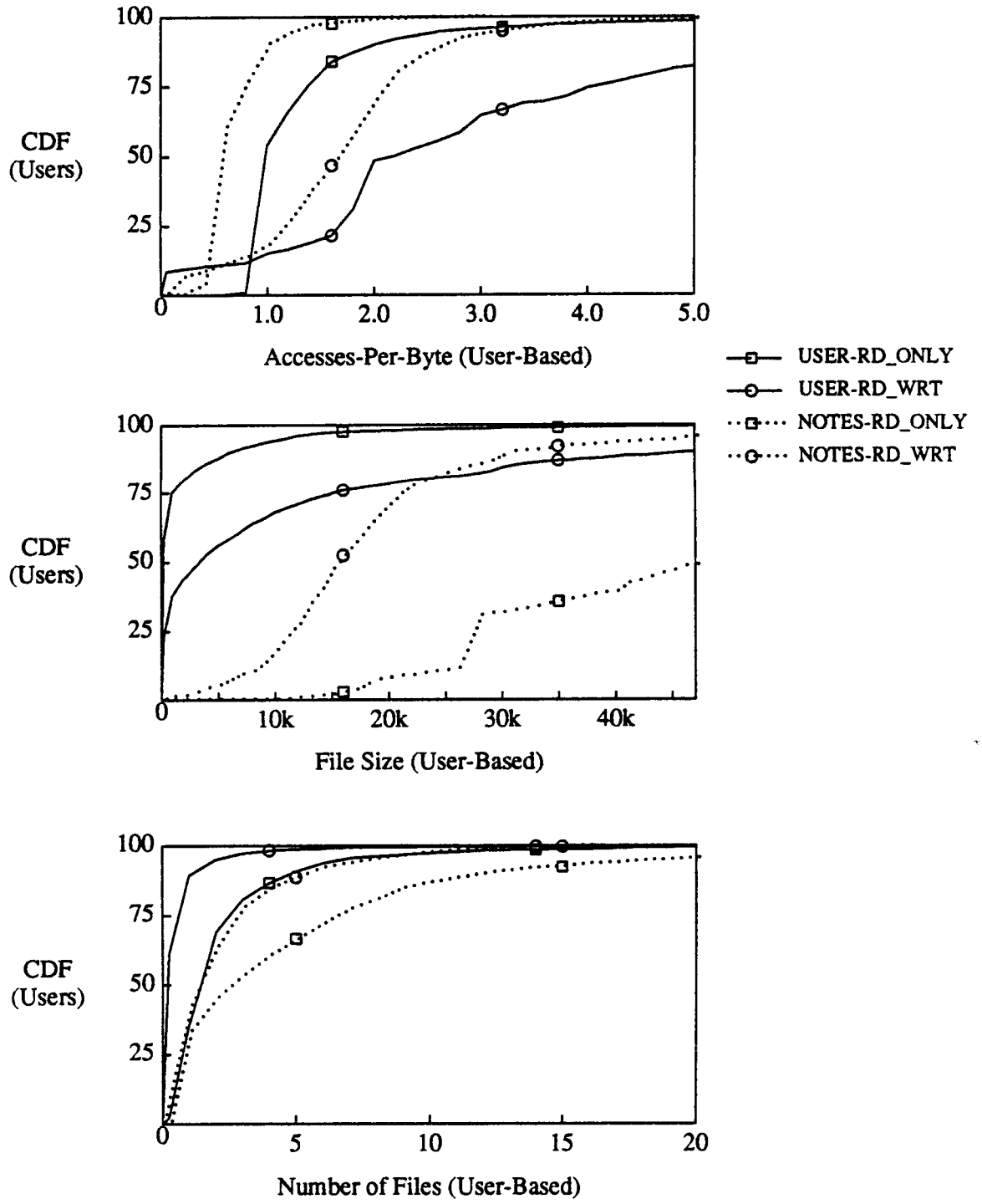


Figure A.1: Distributions of the User Characterization Measures

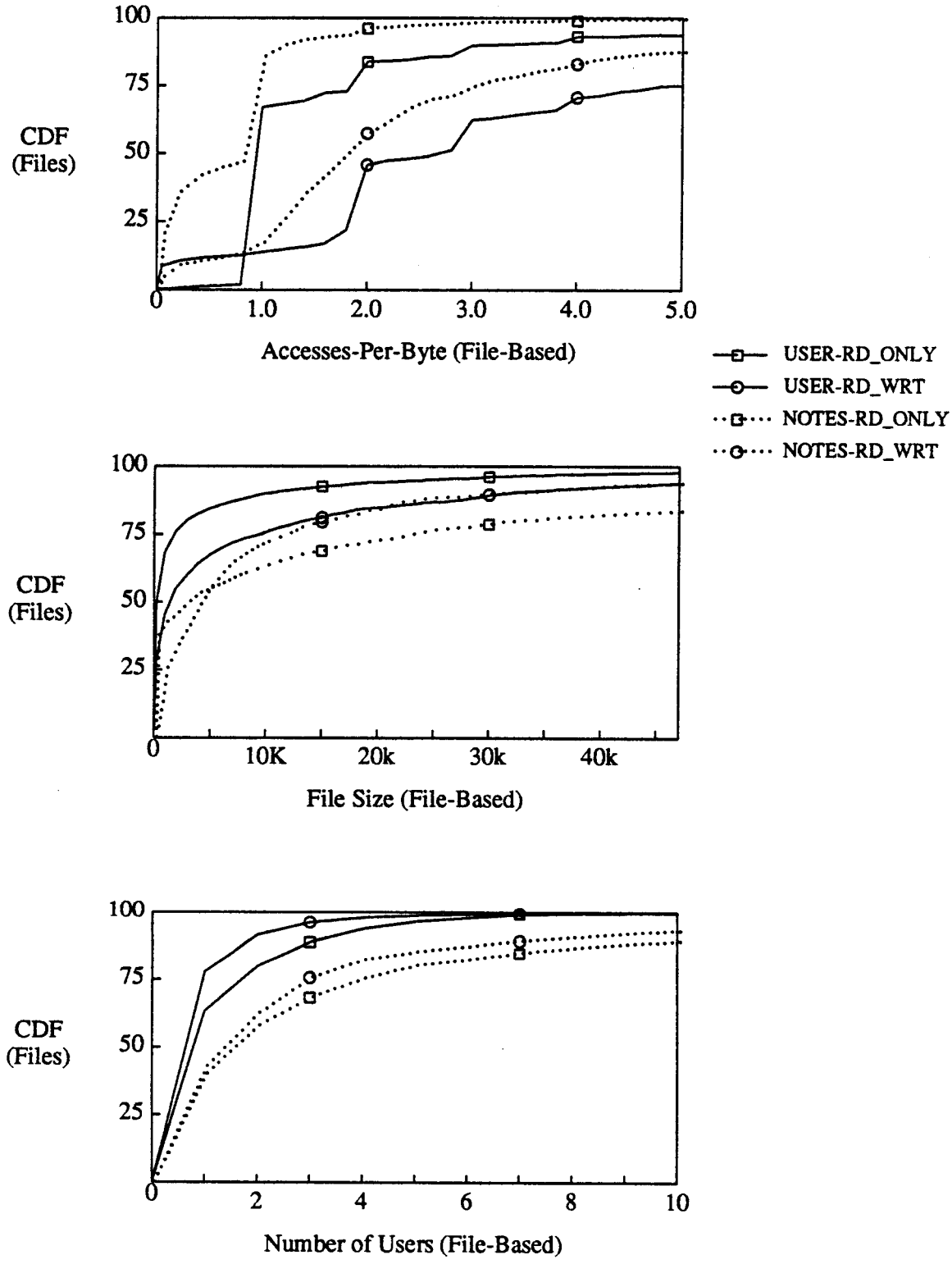


Figure A.2: Distributions of the File Characterization Measures

APPENDIX B

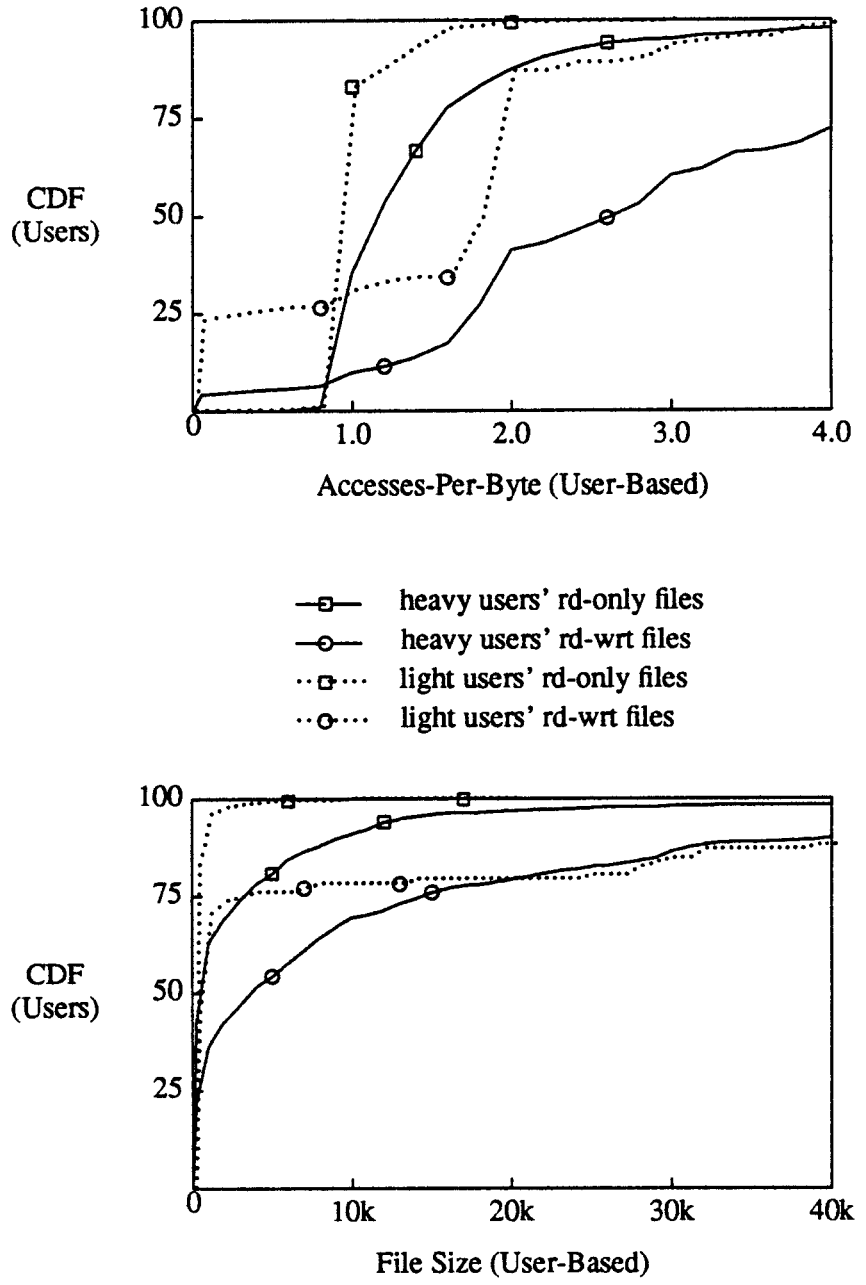


Figure B.1: Distributions of the User Characterization Measures

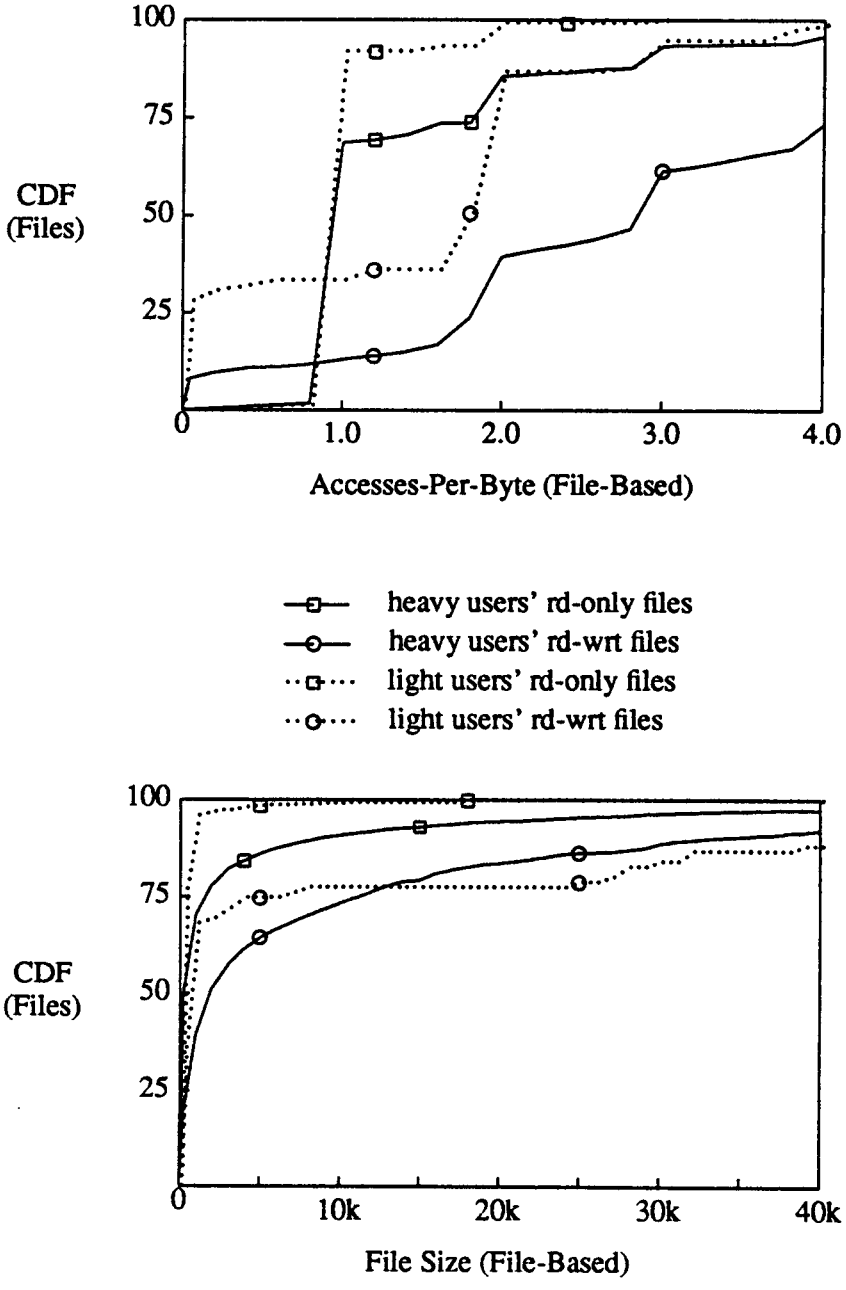


Figure B.2: Distributions of the File Characterization Measures

APPENDIX C

Table C.1: ANOVA models for the user characterization measures and percent sum of square contributions from the factors.

source of variations (factors)	model for accesses-per-byte	model for file size	model for files
file_type	8%	25%	8%
ownership	11%	3%	-
type_of_use	11%	5%	16%
user_type	50%	11%	34%
file_type&ownership	7%	15%	6%
ownership&type_of_use	-	34%	-
file_type&user_type	13%	7%	-
ownership&user_type	-	-	6%
type_of_use&user_type	-	-	30%
R-Square	0.62	0.83	0.85

VITA

Murthy Devarakonda was born in [redacted], on [redacted]. He received his B.E. degree, First Class with Distinction, in Electronics and Communications Engineering from Osmania University, India, in April, 1978. He obtained his M.Tech. degree in Computer Science from Indian Institute of Technology, India, in April, 1980. Before joining the Ph.D. program at the University of Illinois, he obtained his M.S. degree in Computer Science from the University of Wisconsin -- Madison in August, 1983. At the University of Illinois, he was employed as a research assistant from 1983 to 1987.

His current research interests include measurement and performance analysis, file systems, load balancing, and computer networks.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-87-2275 (CSG 79)	5. MONITORING ORGANIZATION REPORT NUMBER(S) NASA
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION NASA
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) NASA Langley Research Center Hampton, VA 23665
8a. NAME OF FUNDING/SPONSORING ORGANIZATION NASA 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NASA-NAG-1-613
8c. ADDRESS (City, State, and ZIP Code) See7b.	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

File Usage Analysis and Resource Usage Prediction: A Measurement-Based Study

12. PERSONAL AUTHOR(S)

Devarakonda, Murthy V.-S.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	1987 December	88

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Unix, measurement, file usage, resource usage, user-behavio statistical modeling, clustering, usage prediction

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In this paper, a probabilistic scheme was developed to predict process resource usage in UNIX. Given the identity of the program being run, the scheme predicts CPU time, file I/O, and memory requirements of a process at the beginning of its life. The scheme uses a state-transition model of the program's resource usage in its past executions for prediction. The states of the model are the resource regions obtained from an off-line cluster analysis of processes run on the system. The proposed method is shown to work on data collected from a VAX 11/780 running 4.3 BSD UNIX. The results show that the predicted values correlate well with the actual. The coefficient of correlation between the predicted and actual values of CPU time is 0.84. Errors in prediction are mostly small. About 82% of errors in CPU time prediction are less than 0.5 standard deviations of process CPU time.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL